Abductive Reasoning on Compliance Monitoring
Balancing Flexibility and Regulation

Federico Chesani[1(✉)], Paola Mello[1(✉)], and Marco Montali[2]

[1] University of Bologna, Bologna, Italy
{federico.chesani,paola.mello}@unibo.it
[2] Free University of Bozen–Bolzano, Bolzano, Italy
montali@inf.unibz.it

Abstract. Many emerging applications in Business Process Management, Clinical Guidelines, Service-Oriented and Multi-Agent Systems, are characterized by distribution, complex interaction and coordination dynamics. Such domains, apparently unrelated, all ask for a suitable tradeoff between flexibility and regulation. In this light, compliance checking emerged as an effective way to understand whether an observed course of interaction agrees with what is expected by a model of the system. In this paper, we single out a non exhaustive list of desiderata and challenges for compliance checking applied at runtime. We then argue that methods, tools and techniques of Computational Logic, and Abductive Reasoning in particular, can be fruitfully exploited to tackle all such challenges in a formally grounded, computationally effective way.

Keywords: Compliance monitoring · Abductive logic programming · Business Process Management · Multi-agent Systems

1 Introduction

Many real world scenarios in the context of the new digital revolution, such as Business Process Management (BPM), e-Health, Web-Services, Multi Agent Systems, Self Adaptive Systems, and Internet of Things and People, require interaction among different entities such as sensors, smart objects, human users, (soft-)bots and APIs. Traces of these interactions can be conceived as streams of data, such as transactional records, logs of process activities, messages produced by multiple, possibly heterogeneous and autonomous sources, and recorded as ordered sets of events. Such data represent a footprint of reality, and their processing provides the basis to understand the relationship between the expected and experienced courses of interaction.

In this light, among the many types of data analysis, of particular interest is *compliance checking*, which compares models and observed events in order to detect possible discrepancies or deviations. Compliance may be assessed at

This is an invited paper, related to the keynote presentation given by Prof. Mello.

M. Kryszkiewicz et al. (Eds.): ISMIS 2017, LNAI 10352, pp. 3–16, 2017.
DOI: 10.1007/978-3-319-60438-1_1

different stages of the lifecycle of a system. In general, though, compliance of complex interacting system cannot be guaranteed by design, nor completely assessed through static verification techniques. On the one hand, it is in fact rarely the case that all interacting components are controlled by a single, centralized orchestrator (think about a service company interacting with external suppliers). On the other hand, compliance typically requires to analyze data that are only available while the system runs (think about a regulation imposing that two tasks have to be executed by the same person). This is why we focus on *runtime compliance checking*, that is, on the problem of assessing compliance of partial, evolving courses of execution.

A plethora of desirable features have been identified in the literature as key towards effective compliance checking. First and foremost, complex scenarios cannot be described by complete and detailed models of interaction. Instead, they call for models that reflect adaptivity to change, and that are able to deal with incomplete information, i.e., models that enjoy *flexibility*. At the same time, interaction has to be disciplined by expressing that the involved entities are expected to behave in agreement with regulations, norms, business rules, protocols and time constraints. This calls for models that incorporate the notion of *regulation*. Beside seeking a suitable tradeoff between flexibility and regulation, other fundamental features have been identified, including support to open and closed models, clear semantics, data and time constraints, and incomplete information.

In this paper, we single out a non exhaustive list of desiderata and challenges for compliance checking applied at runtime. We then show that knowledge representation and reasoning in Artificial Intelligence, and in particular Computational Logic methods based on abduction [32,34], constitute a formally grounded, computationally effective framework to model such complex systems, and provide compliance monitoring facilities. We ground the discussion on the the SCIFF Abductive Framework [6], on the one hand showing how SCIFF integrity constraints capture flexible, expressive interactions models, and on the other hand discussing how the SCIFF abductive proof procedure provides an effective computational mechanism to monitor compliance at runtime.

2 Desired Features for Compliance Frameworks

In compliance checking (see the *Process Mining Manifesto* [2]) a model describing the dynamics of a system is compared with the events related to its concrete executions, to check if the event traces agree with the model's prescriptions.

In an organizational setting, the model is typically constituted by a business process, and its execution traces contain events marking the execution of tasks. Similar abstractions can be found in a plethora of other domains. In multiagent systems, models correspond to interaction protocols, while events are the messages/utterances exchanged by the agents. In healthcare, medical guidelines and clinical pathways play the role of models, while events are represented by the actions performed by the involved healthcare practitioners. In service-oriented

computing, dynamic models of interest are orchestrations/choreographies, while events correspond to service invocations. Finally, in an IoT setting events emerge from sensor and device data, while models aim at guaranteeing global properties that the overall systems should exhibit.

In all these complex scenarios, compliance violations and deviations from what is expected by the model usually require to promptly act, so as to bring the system back within the boundaries defined by the model, plan compensating activities, and/or determine sanctions. This, in turn, calls for runtime compliance checking (or compliance monitoring), in which compliance is assessed over finite, evolving event traces, as defined in [42]. As extensively argued in the literature (see, e.g., [2,42,48]), compliance monitoring poses a number of challenges. We recap some of the most important ones in the remainder of this section.

Trade-off between Flexibility and Compliance to Regulation. Complex scenarios usually cannot be represented by complete and detailed models of interaction, and demands a high level of flexibility to promptly adapt to change. At the same time, interactions must be disciplined so as to ensure that involved entities comply with external regulations, norms, business rules, data and time constraints. In [2] four dimensions are identified, relatively to mining in general, and compliance in particular: (*a*) fitness, (*b*) simplicity, (*c*) precision, and (*d*) generalization. Balancing between such quality dimensions is challenging and highly influences the compliance monitoring quality too. Discrepancies may be interpreted as errors in the observed, or vice-versa as an inadequacy of the model in describing reality (or a combination thereof).

Another approach to tackle flexibility comes from the defeasible interpretation of constraints and norms [28,43]. Requirements can vary from rigid constraints (prescriptions) to soft rules (recommendations). This leads to a non-crisp interpretation of the compliance check, since violation of soft-constraints does not imply non-compliance of the whole, and some constraints may override/defeat other constraints. This is subject of discussion, as pointed out in [20], that advocates the importance of a crisp compliance check: rather than considering violable constraints, new (better) requirements should be elicited.

Real applications often call for exceptions and peculiar situations that could not be directly incorporated in the model without undermining its understandability, nor be ignored. Borrowing a solution from the Object Oriented Programming, we could say that violations should be treated as exceptions, i.e., first class objects/events. Adaptation mechanisms to violations should be supported in the model, and the compliance checking task should take them into account as well.

Open vs Closed Interaction Models. In compliance checking, the model plays a fundamental role in describing the coordination of activities, in terms of required and/or forbidden interactions. Traditionally, procedural models have been adopted for explicitly enumerating strict compliance requirements (e.g., BPMN [31] and Yawl [3] for Business Process, AUML [9] for agents, GLARE [54] for Careflows). Procedural specifications are interpreted as "closed" models, i.e., what is not explicitly cited in the model is forbidden. This is contrasted by declarative, constraint-based models, which adopt an "open" approach, which

supports all courses of execution that satisfy a given set of constraints, compactly and implicitly describing a variety of behaviours. This approach has been proven especially suitable in all those complex scenarios where high flexibility and adaptation are needed, i.e., where procedural approaches would become "spaghetti-like". Notable examples of declarative, open models are the ConDec language for business process management [49], the CLIMB approach to interaction [44], and social commitments in open multiagent systems [51,52,55]). Similar positions can be recognized in the area of programming languages, when Kowalski advocated the need for declarative languages (and for logic programming in particular) in its seminal work "Algorithm = Logic + Control" [35].

Comprehensive Formal Semantics. To a large extent, compliance checking techniques come with ad-hoc implementations that are not backed up by a formal semantics. This poses a twofold problem. First, how compliance is verified is not precisely described, raising concerns about the interpretation of the compliance results. Second, it is not possible to assess the generality of the approach, i.e., if it could be used to deal with compliance in different stages of the system lifecycle, or to solve other problems related to compliance.

E.g., compliance is tightly related to model consistency ("Does the model allows any interaction at all?"), so as to understand whether the model used to assess compliance is faulty. At runtime, compliance could be instrumental to prediction/recommendation ("Given a not-yet completed interaction, what about its possible future courses? What to do next so as to remain compliant?").

Beyond Control Flow. Control-flow aspects (i.e., dynamic constraints among tasks such as sequencing, alternatives, parallel executions [4]) have been extensively studied in the past, also in the context of compliance [50]. However, it is often advocated that more "semantic" constraints shall be considered as well. In [29,43], semantic constraints/rules focus not only on the control flow dimension, but also tackle the data associated to the execution of tasks. The same happens in [41], where structural, data-aware and temporal compliance is tackled. Notice that time-related aspects (such as deadlines, duration constraints, delays, etc.) can be considered a special case data-aware constraints, with the obvious difference that the temporal domain comes with its own specific axioms/properties (e.g., that time flows only forward). Interestingly, approaches coming from the area of computational logic proved particularly useful in the combination of such different ingredients, not only to handle compliance monitoring [45,46], but also to deal with other forms of reasoning and verification [27,53].

Mixing Procedural and Declarative Aspects. Works like [29,43] argue that properties related to data and time do not always fit well with control flow modelling. Consider the following general rule applied in medicine: "Do not administer a certain drug if the patient is allergic to any of its components". This rule applies to each step in a process execution, and would not be well captured in the control flow description of the process. Generally speaking, this is the problem of suitably mixing procedural and declarative knowledge, and it emerges also in the legal setting when it comes to so-called *regulatory compliance*

[10]. Logic-based frameworks, such as [12,28,29], show that these two seemingly contrasting approaches can be captured in a coherent and unified way.

Missing Information. Information about the model and the execution can be incomplete; some events can be missing (or not observable) or described in a partial and/or noisy, and/or uncertain way. Hence, the degree of discrepancy between the interaction model and the execution becomes hard to be evaluated, and compliance checking can be a difficult hard task.

Also temporal information might be affected by incompleteness, with consequences like, for example, being difficult to predict the duration of the tasks, as well as to check inter-tasks temporal constraints. In [21], for example, an ad-hoc algorithm is proposed to ensure dynamic controllability of a workflow (i.e., there exists at least one way to ensure the compliance of a workflow, independently of some task and their unknown duration).

Extraction of Meaningful Events. Complex systems can provide a huge number of low-level, meaningless information. Two issues emerge here. The first concerns the conceptual granularity of such events: quite often, meaningful events (w.r.t. compliance) are not traceable, but need to be derived by properly aggregating the low-level, recorded data. This problem is connected to the more general problem of *activity/event recognition* (cf., e.g., [8]). The second issue concerns the manipulation of large-scale streams of low-level events, which falls under the umbrella of *complex event processing* [40].

Scalability of Compliance Monitoring. Issues arise when compliance needs to be assessed on large systems generating "big event data". In [1], van der Aalst argues that scalability issues can be tamed through horizontal and/or vertical partitioning, so as to distribute the compliance monitoring task over a network of nodes (e.g. multi-core systems, grid infrastructures, or cloud environments).

3 Abductive Logic Programming in Action

We now introduce the paradigm of abductive logic programming, and show how it can be used to reason on compliance, obtaining a general, coherent framework in which all aspects mentioned in Sect. 2 can be suitably tackled.

3.1 Abductive Logic Programming

Abduction is a non-monotonic reasoning process, where facts are observed, and hypotheses are made to explain the observed facts [33]. Given a Logic Program $\mathcal{P}$ and an observation $\mathcal{G}$ (the goal), deductive reasoning tries to prove that $\mathcal{G}$ is a logic consequence of $\mathcal{P}$. However, this might not be possible due to some missing knowledge in $\mathcal{P}$. Abduction tackles this issue by looking for a set Δ of hypotheses such that $(\mathcal{P} \cup \Delta)$ logically entails the goal. An abductive framework provides the reasoning tool for formulating such hypotheses.

Typically, not all configurations of hypotheses make sense. To declaratively capture how hypotheses relate to each other, *integrity constraints* are employed.

A typical integrity constraint (IC) is a *denial*, expressing that two explanations are mutually exclusive. Given a set $\mathcal{IC}$ of integrity constraints, an abductive explanation Δ must be such that $(\mathcal{P} \cup \Delta)$ logical entails all ICs present in $\mathcal{IC}$.

Abductive Logic Programs (ALP) have been formalized in [32]. There, an ALP is defined as a triple $\langle \mathcal{KB}, \mathcal{A}, \mathcal{IC} \rangle$, where: *(i)* $\mathcal{KB}$ is a logic program, *(ii)* $\mathcal{A}$ is a set of abducible predicates, and *(iii)* $\mathcal{IC}$ a set of ICs. Given a goal $\mathcal{G}$, abductive reasoning looks for a set of literals $\Delta \subseteq \mathcal{A}$ such that they entail $\mathcal{G} \cup \mathcal{IC}$. The integration of constraint solving (ACLP) [22,32] enhances the practical utility of ALP by enriching the representation of the problem domain, and by empowering the computation of abductive explanations.

3.2 The SCIFF Framework

SCIFF [6] is a ACLP framework that extends Fung and Kowalski's IFF proof-procedure for ALP [26] towards compliance checking. Conceptually, this is done by introducing two types of special predicates, *happened events* and *expectations*.

A *happened event*, denoted by $\mathbf{H}(Ev, T)$, captures the idea that event Ev occurs at timestamp T. Ev is a logical term, and consequently may refer to a complex description of the event and its data. For example, fact

$$\mathbf{H}(administer\_drug(p42, paracetamol, mg, 500), 45).$$

indicates that at time 45, 500 mg of paracetamol have been given to the patient $p42$. A finite set of facts denoting happened events constitutes a *trace*.

Expectations capture the expected behaviour, and come with a positive or a negative flavour. A *positive expectation* $\mathbf{E}(Ev, T)$ states that event Ev is expected to happen at time T. Variables employed therein are quantified existentially. A *negative expectation* $\mathbf{EN}(Ev, T)$ states that event Ev is expected not to happen at time T. Variables employed therein are quantified universally. For example

$$\mathbf{E}(administer\_drug(p34, paracetamol, mg, 1000), T_1) \wedge T_1 < 78$$
$$\wedge\, \mathbf{EN}(administer\_drug(p67, paracetamol, mg, Q), T_2).$$

models that two expectations are in place. First, patient $p34$ is expected to receive 1 g of paracetamol within the deadline of 78. Second, patient $p67$ is expected to not receive any quantity of paracetamol at any time.

In SCIFF, models are specified through three components: (i) a knowledge base $\mathcal{KB}$, i.e. a set of (backward) rules $head \leftarrow body$; (ii) a set $\mathcal{IC}$ of ICs, each of which is a (forward) rule of the form $body \rightarrow head$; and (iii) a goal $\mathcal{G}$ (as in logic programming). The integrity constraints can be considered as reactive rules, i.e., when the body becomes true (because of the occurrence of the involved events), then the rule "fires" and the expectations in the head are generated. E.g., IC

$$\mathbf{H}(temperature(P, Temp), T_1) \;\wedge\; Temp > 38 \;\rightarrow$$
$$\mathbf{E}(administer\_drug(P, paracetamol, mg, 1000), T_2) \wedge T_2 > T_1 \wedge T_2 < T_1 + 10.$$

states that if a patient P is detected to have the temperature at a time T_1, then it is expected that P is given 1 g paracetamol, within ten time units after T_1.

Intuitively, SCIFF defines compliance as a sort of "hypothesis" confirmation relating happened events and expectations: a trace is compliant with a SCIFF specification if all expectations generated by combining the trace with the ICs are confirmed, i.e., for every positive expectation (**E**) there is a *corresponding* happened event (**H**) in the trace, and if for every negative expectation (**EN**) there is no *corresponding* happened event (**H**) in the trace. The *correspondence* between events and expectations is grounded on the notion of *unification* in Logic Programming [38]. Finally, SCIFF is inherently "open": everything that is not explicitly forbidden is indeed allowed. The interested reader might refer to [6] for a comprehensive description of the SCIFF Framework.

3.3 On the Suitability of SCIFF to Monitor Compliance

We now go through the different desired features for compliance frameworks discussed in Sect. 2, and argue how they can be tackled using SCIFF.

Formal Semantics, Flexibility, Open/Closed Models. Being SCIFF grounded on ALP, it natively comes with a formal declarative semantics, and with a corresponding (sound and complete) proof procedure to generate and confirm expectations. Furthermore, the notion of expectation confirmation briefly recalled in Sect. 3.2 allows us to lift the semantics of ALP to handle (runtime and offline) compliance [44]. Other reasoning tasks, such as model consistency and prediction/recommendation, can be supported as well (see our preliminary works in [14,15]). The definition of compliance rules in terms of logic-based specifications also allowed us to learn rules from traces, by realizing a form of discriminative mining through inductive logic programming techniques [16].

Notice that a number of concrete, end-user oriented languages for interaction/process modeling can be translated into SCIFF, and consequently directly inherit such compliance-related functionalities. E.g., we translated into SCIFF the constraint-based service interaction language DecSerFlow [47], and the AUML language for multiagent interaction protocols [5].

Notably, DecSerFlow and AUML are radically different in the way they approach interaction models, since they respectively adopt an open and closed approach to interaction. In Sect. 3.2, we argued that SCIFF adopts an open approach, so one might wonder how is it possible to capture closed models in SCIFF. We show this versatility by considering the sequencing between two events a and b, that is, the compliance rule indicating that whenever a occurs, then b should occur afterwards. In an open setting, any other event may happen before, during and after the sequence. The SCIFF formalization of this open interpretation is constituted by the IC:

$$\mathbf{H}(a, T_a) \rightarrow \mathbf{E}(b, T_b) \wedge T_b > T_a.$$

What if, instead, one would prefer a closed interpretation of the sequence, that a and b should occur next to each other? This could be declaratively captured by suitably exploiting temporal constraints, negative expectations, and logic programming variables in the following two ICs:

$$\mathbf{H}(a, T_a) \rightarrow \mathbf{E}(b, T_b) \wedge T_b > T_a \wedge \mathbf{EN}(X, T_x) \wedge T_a < T_x < T_b.$$
$$\mathbf{H}(b, T_b) \rightarrow \mathbf{E}(a, T_a) \wedge T_b > T_a \wedge \mathbf{EN}(X, T_x) \wedge T_a < T_x < T_b.$$

The first IC models again the sequence, but it also forbids the occurrence of any event between a and b. The second IC instead completes the notion of "chaining" between a and b, indicating that if b occurs, a should have occurred previously.

This modelling technique can be tuned so as to capture hybrid models combining openness and closeness. Finally, a generalized notion of closeness may also be realized by revisiting the declarative semantics of compliance provided by SCIFF, in particular expressing that a trace is compliant if *and only if* for every happened events there is a corresponding expectation, *and vice-versa*.

Beyond Control Flow. Since SCIFF is based on (constraint) logic programming, it inherits the expressive power needed to capture background, domain knowledge in the form of a knowledge base. In addition, events may be represented using complex, structured terms involving data, and data-aware constraints are seamlessly supported as constraints over variables used inside such terms. As shown above, special variables are used to explicitly denote timestamps, consequently allowing one to specify both qualitative (e.g., response, precedence, sequencing) and quantitative (e.g., delays, deadlines) temporal conditions by means of constraints involving such special variables.

Mixing Procedural and Declarative Aspects. SCIFF naturally lends itself to model declarative compliance rules, but procedural flow constructs can be captured as well [15,17]. Unfortunately, a direct combination of declarative and procedural specifications is not always satisfactory, since the two may impose conflicting requirements. For example, in the healthcare domain procedural guidelines are used to indicate a default treatment, and subtly interact with the background medical knowledge used to avoid harmful situations. E.g., a guideline for treating pneumonia may prescribe the administration of penicillin, which would conflict with the general clinical practice if the patient is allergic to penicillin. The conflict is resolved, in this case, by administering a different drug with equivalent effects. The mindful reader might object that this interaction can be tackled by explicitly incorporating alternative choices within the guideline. However, this approach would hinder the readability of the guideline, and may become unmanageable in general, since it requires to foresee, at modelling time, all possible subtle interactions between these two sources of knowledge.

A suitable way to manage this types of conflict while retaining understandability and flexibility is an open challenge. In [11] we separated the representation of the guideline and the basic medical knowledge, and assumed a user-defined priority of the medical knowledge with respect to the guideline prescriptions. This solution is peculiar to the medical field, and cannot be directly generalized. Relevant directions of research in this respect are approaches that assign different weights to rules, like defeasible logic, default logic, and variants of preference logic [7,30].

Missing Information. In a number of applications the collection of event data is far from being precise and complete: some events may not be monitorable, or

may be monitored by error-prone systems, thus producing incomplete, corrupted and/or noisy traces. Such common situations require to suitably revise the notion of compliance, so as to distinguish noncompliant traces from traces that, due to logging issues, appear to be noncompliant. Thanks to its abductive semantics, SCIFF naturally lends itself to deal with this setting, as shown in [14,15]. There, a novel approach to compliance is presented, where beside yes/no answer, also the concept of weak compliance is explored. The idea is that a trace with missing information might be weak compliant with respect to a SCIFF specification, if it is possible to make a set of consistent hypotheses about the missing information. This approach simultaneously accounts for missing events, events with missing data (e.g., a logged event without the indication of "who" generated it), and events with missing time. Intuitively, missing events are handled by allowing SCIFF to hypothesise the occurrence of expected events, and to assess weak compliance by checking whether the overall set of formulated hypotheses is consistent. Traces with incomplete information are instead dealt with by allowing the happened events contained therein to include variables, and letting SCIFF formulating hypotheses on the values that such variables may take.

Of course, the problem of incomplete information in compliance may be tackled with other techniques, such as planning [25], and would also benefit from techniques that are able to tame uncertainty, such as fuzzy set theory, probabilistic reasoning, evolutionary computing, and machine learning.

Extraction of Meaningful Events. A well-established approach to complex event/activity recognition and processing in a declarative setting is that of the Event Calculus (EC) [36]. EC is a powerful logic-based framework to reason on the effects that events have on the "state" of the world. On the one hand, this mechanism can be used to infer high-level events from low level samples [8]. On the other hand, it can be exploited to reason on the effects that events have on the normative state of business interactions, which in turn can be used to relate compliance at the level of events with compliance at the level of contracts and commitments [19,55].

Notably, in [18] it has been shown that the EC can be formalized in SCIFF, with a twofold advantage. From the semantical point of view, event-based and state-based specifications are reconciled in a single logical framework. From the operational point of view, the SCIFF proof procedure provides a very effective form of runtime reactive reasoning for the EC, which is quite unique in the literature.

Scalability of Compliance Monitoring. The expressiveness of SCIFF, and the sophistication of its proof procedure, make it sensible to performance and scalability issues. An open challenge concerns the exploitation of modern parallelization and distribution architectures so as to improve the efficiency of compliance monitoring in SCIFF.

Interesting insights towards this goal come from the BPM field, where [1] has proposed an approach based on horizontal and vertical partitioning, to split traces and models and decompose the evaluation of compliance accordingly. Specifically, vertical partition assumes the existence of a special data slot,

called *case identifier*, that is attached to all happened events and that is used to correlate and group them into different traces depending on the case. As an example, consider a loan management process: while the information system used to handle loans would log all events about all requested loans, such events may be separated into different traces on the basis of the loan identifier. This impacts on compliance, as each trace can be combined with the regulatory model of interest independently form the other traces, paving the way towards a direct parallelization/distribution of compliance monitoring.

In horizontal partitioning the regulatory model and the monitored traces are decomposed into sub-models and sub-traces, on the basis of the relevance of events to the different sub-models. This technique is effective if the overall framework guarantees some *compositionality* property, that is, if compliance of a trace with a regulatory model can be simply derived by checking compliance of its sub-traces with the corresponding sub-models. In general, the determination of sub-models and sub-traces so as to preserve compositionality is a difficult task, especially in the case of procedural models, while logic-based approaches naturally lend themselves to be decomposed.

Vertical and horizontal partitioning techniques have been reconstructed within SCIFF in [39], paving the way towards compliance monitoring via Map-Reduce. A similar approach has been investigated in [13], for the context of multiagent systems.

4 Conclusions

In this overview paper, we have introduced the important problem of compliance checking, focusing in particular on runtime compliance monitoring in the context of several application domains. We have argued that abductive reasoning, and in particular the SCIFF abductive logic programming framework, provides a comprehensive and rich approach to model sophisticated, flexible regulatory specifications, and to operationally monitor compliance. We have also outlined a number of challenges, how they can be tackled in SCIFF, and how they open interesting lines of research for the future.

It is worth recalling that checking compliance in SCIFF amounts to formulate hypotheses and abductive explanations that provide very valuable insights to human stakeholders. In this respect, this approach is fully in line with the recent debate on AI as “artificial” vs “augmented” intelligence.

Two additional open challenges are worth to be mentioned when it comes to aiding humans in compliance assessment. A first crucial challenge concerns how to aggregate and combine multiple alternative hypotheses formulated by SCIFF when monitoring compliance, so as to obtain a sort of integrated, compact explanation for the essential causes for noncompliance. Specifically, we are interested in understanding whether it is possible, in the context of compliance, to reconstruct a notion similar to that of *event structure* [23], which has been advocated as a unifying representation formalism for processing models and traces.

A second important challenge is how to exploit the formal foundations provided by SCIFF so as to go beyond compliance checking. As witnessed by recent

works [24,37], there is a great interest towards more advanced techniques to estimate the "distance" between a regulatory model and a monitored trace, possibly also indicating how to realign/repair the trace and/or the model so as to restore compliance.

References

1. van der Aalst, W.M.P.: Distributed process discovery and conformance checking. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 1–25. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28872-2_1
2. van der Aalst, W.M.P., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28108-2_19
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005)
4. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
5. Alberti, M., Chesani, F., Daolio, D., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interaction protocols in a logic-based system. Scalable Comput.: Pract. Exp. **8**(1), 1–13 (2007)
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Trans. Comput. Log. **9**(4), 29:1–29:43 (2008)
7. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Trans. Comput. Log. **2**(2), 255–287 (2001)
8. Artikis, A., Sergot, M.J., Paliouras, G.: An event calculus for event recognition. IEEE Trans. Knowl. Data Eng. **27**(4), 895–908 (2015)
9. Bauer, B., Cossentino, M., Cranefield, S., Huget, M.P., Kearney, K., Levy, R., Nodine, M., Odell, J., Cervenka, R., Turci, P., Zhu, H.: The FIPA Agent UML. http://www.auml.org/
10. Boella, G., Janssen, M., Hulstijn, J., Humphreys, L., van der Torre, L.W.N.: Managing legal interpretation in regulatory compliance. In: ICAIL 2013, pp. 23–32. ACM (2013)
11. Bottrighi, A., Chesani, F., Mello, P., Montali, M., Montani, S., Terenziani, P.: Conformance checking of executed clinical guidelines in presence of basic medical knowledge. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 100, pp. 200–211. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28115-0_20
12. Bragaglia, S., Chesani, F., Mello, P., Montali, M.: Conformance verification of clinical guidelines in presence of computerized and human-enhanced processes. In: Hommersom, A., Lucas, P.J.F. (eds.) Foundations of Biomedical Knowledge Representation. LNCS, vol. 9521, pp. 81–106. Springer, Cham (2015). doi:10.1007/978-3-319-28007-3_6
13. Briola, D., Mascardi, V., Ancona, D.: Distributed runtime verification of JADE and Jason multiagent systems with prolog. In: CILC 2014, CEUR, vol. 1195, pp. 319–323 (2014)
14. Chesani, F., De Masellis, R., Di Francescomarino, C., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Abducing compliance of incomplete event logs. In: Adorni, G., Cagnoni, S., Gori, M., Maratea, M. (eds.) AI*IA 2016. LNCS, vol. 10037, pp. 208–222. Springer, Cham (2016). doi:10.1007/978-3-319-49130-1_16

15. Chesani, F., De Masellis, R., Francescomarino, C.D., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Abducing workflow traces: a general framework to manage incompleteness in business processes. In: ECAI 2016. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1734–1735. IOS Press (2016)
16. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. In: Jensen, K., Aalst, W.M.P. (eds.) Transactions on Petri Nets and Other Models of Concurrency II. LNCS, vol. 5460, pp. 278–295. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00899-3_16
17. Chesani, F., Mello, P., Montali, M., Storari, S.: Testing careflow process execution conformance by translating a graphical language to computational logic. In: Bellazzi, R., Abu-Hanna, A., Hunter, J. (eds.) AIME 2007. LNCS, vol. 4594, pp. 479–488. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73599-1_64
18. Chesani, F., Mello, P., Montali, M., Torroni, P.: A logic-based, reactive calculus of events. Fundam. Inform. **105**(1–2), 135–161 (2010)
19. Chesani, F., Mello, P., Montali, M., Torroni, P.: Representing and monitoring social commitments using the event calculus. Auton. Agents Multi-Agent Syst. **27**(1), 85–130 (2013)
20. Chopra, A.K.: Requirements-driven adaptation: compliance, context, uncertainty, and systems. In: RE@RunTime 2011, pp. 32–36. IEEE (2011)
21. Combi, C., Posenato, R.: Towards temporal controllabilities for workflow schemata. In: TIME 2010, pp. 129–136. IEEE (2010)
22. Denecker, M., Kakas, A.: Abduction in logic programming. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2407, pp. 402–436. Springer, Heidelberg (2002). doi:10.1007/3-540-45628-7_16
23. Dumas, M., García-Bañuelos, L.: Process mining reloaded: event structures as a unified representation of process models and event logs. In: Devillers, R., Valmari, A. (eds.) PETRI NETS 2015. LNCS, vol. 9115, pp. 33–48. Springer, Cham (2015). doi:10.1007/978-3-319-19488-2_2
24. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
25. Francescomarino, C., Ghidini, C., Tessaris, S., Sandoval, I.V.: Completing workflow traces using action languages. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 314–330. Springer, Cham (2015). doi:10.1007/978-3-319-19069-3_20
26. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. J. Log. Program. **33**(2), 151–165 (1997)
27. Giordano, L., Martelli, A., Spiotta, M., Dupré, D.T.: Business process verification with constraint temporal answer set programming. TPLP **13**(4–5), 641–655 (2013)
28. Governatori, G., Hashmi, M., Lam, H.-P., Villata, S., Palmirani, M.: Semantic business process regulatory compliance checking using LegalRuleML. In: Blomqvist, E., Ciancarini, P., Poggi, F., Vitali, F. (eds.) EKAW 2016. LNCS, vol. 10024, pp. 746–761. Springer, Cham (2016). doi:10.1007/978-3-319-49004-5_48
29. Governatori, G., Rotolo, A.: Norm compliance in business process modeling. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 194–209. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16289-3_17
30. Greco, S., Trubitsyna, I., Zumpano, E.: On the semantics of logic programs with preferences. J. Artif. Intell. Res. (JAIR) **30**, 501–523 (2007)
31. Initiative, B.P.M.: Business process modeling notation. http://www.bpmn.org
32. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992). http://dx.doi.org/10.1093/logcom/2.6.719

33. Kakas, A.C., Mancarella, P.: Abduction and abductive logic programming. In: Logic Programming, Proceedings of ICPL, pp. 18–19 (1994)
34. Kakas, A.C., Michael, A., Mourlas, C.: ACLP: abductive constraint logic programming. J. Log. Program. **44**(1–3), 129–177 (2000)
35. Kowalski, R.A.: Algorithm = logic + control. Commun. ACM **22**(7), 424–436 (1979)
36. Kowalski, R.A., Sergot, M.J.: A logic-based calculus of events. New Gener. Comput. **4**(1), 67–95 (1986)
37. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. Inf. Syst. **47**, 258–277 (2015)
38. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Berlin (1987)
39. Loreti, D., Chesani, F., Ciampolini, A., Mello, P.: Distributed compliance monitoring of business processes over mapreduce architectures. In: ICPE 2017, to appear
40. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Boston (2001)
41. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., Dadam, P.: SeaFlows toolset – compliance verification made easy for process-aware information systems. In: Soffer, P., Proper, E. (eds.) CAiSE Forum 2010. LNBIP, vol. 72, pp. 76–91. Springer, Heidelberg (2011). doi:10.1007/978-3-642-17722-4_6
42. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
43. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Inf. Syst. Front. **14**(2), 195–219 (2012)
44. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach. LNBIP, vol. 56. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14538-4
45. Montali, M., Chesani, F., Mello, P., Maggi, F.M.: Towards data-aware constraints in declare. In: Proceedings of SAC 2013. ACM (2013)
46. Montali, M., Maggi, F.M., Chesani, F., Mello, P., van der Aalst, W.M.P.: Monitoring business constraints with the event calculus. ACM TIST **5**(1), 17:1–17:30 (2013)
47. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. TWEB **4**(1), 3:1–3:62 (2010)
48. Munoz-Gama, J.: Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. LNBIP, vol. 270. Springer, Heidelberg (2016). doi:10.1007/978-3-319-49451-7
49. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). doi:10.1007/11837862_18
50. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
51. Singh, M.P.: Agent communication languages: rethinking the principles. In: Huget, M.-P. (ed.) Communication in Multiagent Systems. LNCS, vol. 2650, pp. 37–50. Springer, Heidelberg (2003). doi:10.1007/978-3-540-44972-0_2
52. Singh, M.P., Chopra, A.K., Desai, N.: Commitment-based service-oriented architecture. IEEE Comput. **42**(11), 72–79 (2009)

53. Smith, F., Proietti, M.: Rule-based behavioral reasoning on semantic business processes. In: ICAART 2013. SciTePress (2013)
54. Terenziani, P., Raviola, P., Bruschi, O., Torchio, M., Marzuoli, M., Molino, G.: Representing knowledge levels in clinical guidelines. In: Horn, W., Shahar, Y., Lindberg, G., Andreassen, S., Wyatt, J. (eds.) AIMDM 1999. LNCS, vol. 1620, pp. 254–258. Springer, Heidelberg (1999). doi:10.1007/3-540-48720-4_28
55. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 235–247. Springer, Heidelberg (2002). doi:10.1007/3-540-45448-9_17