Declarative Approaches for Compliance by Design

Francesco Olivieri[1](✉), Guido Governatori[1], Nick van Beest[1], and Nina Ghanbari Ghooshchi[2]

[1] Data61, CSIRO, Dutton Park, Brisbane, Australia
{francesco.olivieri,guido.governatori,nick.vanbeest}@data61.csiro.au
[2] Griffith University, Brisbane, Australia
nina.ghanbari@griffithuni.edu.au

Abstract. The interest of scholars in devising automated methods to describe and analyse business processes has increased in the last decades due to the extreme interest of organisations in achieving their business objectives while remaining compliant with the relevant normative system. Adhering with norms and policies does not only help to avoid severe sanctions but also results in greater confidence by the consumers, and prestige for the organisation. Defining processes through the paradigm of *declarative specifications* is gaining momentum due to its intrinsic characteristic of being able to capture business as well as normative specifications within the same framework. We describe some of the state of the art techniques in the field of Business Process Compliance, focusing on pros and cons of such techniques, and advancing future lines of research.

1 Introduction

Business processes are used world-wide by organisations at every hierarchical level for diverse purposes. We can identify two causative reasons. First, they provide a good source of information about the activities and capabilities of an organisation. Second, such information is used to improve them. Business Process Management (BPM) can be described as a "process optimisation process". Being a holistic managerial approach, BPM considers processes as strategic means of an organisation that must be understood, analysed, and improved to continually furnish better and increasingly desirable products to clients. These processes are critical to any organisation as they often represent a significant proportion of costs.

For the benefits brought by BPM to be effective, suitable representations of business processes should be given. While an experienced programmer writes thousands of lines of code, a typical user (or process owner) does not want, or have the ability, to analyse complicated or convoluted formulas. They instead want simple, easy to understand representations. In this sense, Business Process Modelling technology emerged as a strong paradigm for the modelling, analysis, improvement, and automation of the day-to-day activities of organisations. The field is now a mature research area with widespread industry adoption. Business Process Modelling covers a wide variety of methodologies: from graphical modelling languages to ease the understanding of the stakeholders (e.g., YAWL [33],

A. Beheshti et al. (Eds.): ASSRI 2015/2017, LNBIP 234, pp. 80–97, 2018.
https://doi.org/10.1007/978-3-319-76587-7_6

EPC [38], BPMN[1]) to fully precise mathematical formalisms (e.g., Petri nets [37], π-calculus [24]) for formal analysis and automated process verification.

All the above mentioned formalisms and representations fall into the family of *imperative approaches*: they define a process model as a detailed specification of a step-by-step procedure that should be followed during the whole execution. In such a way, they strictly specify how the process will be executed. If from one side this procedural nature is their strength, it is also their main drawback. In fact, they suffer from some limitations. First, it is sometimes hard to obtain precise information about the order of the actions to be performed from the business requirements. Second, such a paradigm is not suitable to capture flexible business processes, i.e., processes whose internal structure and relationships among the various tasks is dynamic and with a large degree of variations (e.g., triage processes in hospital emergency rooms). Third, their imperative nature yields over-specified and highly-structured processes [39] where it is difficult to define relationships among the atoms. For instance, it is possible to model a simple statement as "activities A and B should never occur together" only through a detailed strategy to implement it.

In the opposing direction moves the school of modelling processes by *declarative specifications* [5,14,30]. Instead of specifying a process step by step, the focus in this approach being on defining relationships among the tasks to be executed to achieve a goal, as well as in understanding the behaviour of such "atoms". By shifting the focus from the whole process to its basic building blocks, you gain knowledge regarding which preconditions trigger the activation of a task (*inputs*), as well as what happens once a task completes its execution (*outputs*). It is indeed a common practice that organisations develop business rules manuals for their operations: such business rules may specify constraints that apply to their business processes (e.g., a customer has to be older than 18 in order to be eligible for a loan). As organisations grow, so do their processes and business rules. As a consequence, the number of business rules is generally very large [25].

Another important value of the declarative specification approach is that we can combine business specifications with normative specifications within a single framework. This is a crucial aspect of BPM for two reasons. From one side, the field of Business Process Compliance studies that the business practices are not in breach with the legislation regulating the organisational environment [19,34]. Worldwide scandals such as Societe Generale (France), Enron (USA) and HIH (Australia) forced governments and standard organisations to enact more restrictive regulatory mandates leading enterprises to massive investments in the market of compliance related software and services (over \$30billion in 2008 [20]). Scholars have studied automated methods to establish whether a business process is compliant or not with the norms ruling the environment where the organisation acts in [16–18] and BPC deals with the problem of developing the above mentioned methods.

Secondly, compliance requirements are usually formulated as a set of rules that can be checked during, or after, the execution of the business process,

[1] http://www.bpmn.org.

called compliance by detection. If a non-compliant behaviour is detected, the business process needs to be redesigned. Alternatively, the rules can already be taken into account while modelling the business process. The result is a business process that is compliant by design. This technique, which goes under the name of *compliance by design*, has the advantage that a subsequent verification of compliance is not required. Automated tools able to generate *compliant by design* processes have some clear advantages: (i) being a preventative methodology, a subsequent compliance verification is not needed, (ii) it is possible to analyse all possible execution paths within the rules, (iii) the generated business process is optimised for execution of the business rules and regulations, as it is specifically designed to exactly represent the behaviour allowed by the rules.

Let us consider which challenges such automated tools need to address. First, we need a formalism able to represent in a coherent, functional, and possibly compact manner the business rules, the organisational objectives, and the normative system. Moreover, the framework should be able to determine whether a particular objective is attainable without violating the relevant norms in a given scenario, and which tasks are involved in this process. Thus, the deliberation effectively generates a plan. A question may arise: why a logical formalism is suitable to represent (business) processes? The *derivation* (or *formal proof*) of a statement is the final phase of a finite sequence of sentences/steps each of which is a fact (a statement that is given as a truth), or follows from the preceding sentences in the sequence by the application of a *rule*. A typical rule consists of a *set of preconditions* (*antecedents*) and some *conclusions* (*postconditions*). Whenever such preconditions are satisfied, the rule is enabled and produces its conclusion; absent the preconditions the action cannot be taken and, if it is taken, the postconditions hold. As such, a derivation has a strong, semantical correspondence with a trace of a process, and we can hence establish a bijection between a process and a logic theory. This is in line with the definition of (business) process: a task is the result of the successful execution of previous tasks (preconditions) and, in turn, may take part in the activation of one or more other tasks. This mechanism fully captures the idea of control flow in terms of satisfiability over a set of formalised constraints: each derivation can be seen a simulation of an *execution trace*. The logical apparatus we take into account is the one proposed in [14,15].

The second challenge lies in the *extraction* of the actual process from the above logical description: we need to "put together" such information to obtain a structured process, i.e., a process where the tasks in the traces are structured in sequential, parallel and alternative patterns. To the extent of our knowledge, two approaches were most successful. The former [28,29] lies within the field of representing business processes through Business Process Model Notation. The latter adopts Petri nets for their intrinsic characteristic of permit a direct formal verification of the net (process) [12].

Our agenda is as follows. We start with Sect. 2, where we give a more detailed description about *capabilities rules* and why they are suitable to represent tasks and control flow. Follows Sect. 3: in there, we introduce the modal, skeptical logics which is able to represent actions, norms and goals. Section 4 describes two different approaches to visualise and operationalise such sets of rules as a verifiable business process. In Sect. 5 we discuss pros and cons of the two proposed methods; the related work follows in Sect. 6. We end the paper with Sect. 7.

2 Rules for Declarative Processes

Governatori *et al.* [14,15] proposed an agent-oriented rule language for the declarative specifications of *norm* and *goal compliant* business processes. The main idea is that the set of rules can be partitioned into three subsets: a set of rules describing the "capabilities" of an organisation, a set of rules corresponding to the norms governing a process, and a set of rules encoding the objectives/goals of an organisation to fulfil in their processes. The intuition behind the *capability rules* is that they model the set of activities/tasks an organisation is able to carry out, the preconditions required for each task, the effects of executing such tasks, and the relationships among them. The language upon which the rules are defined consists of a set of two types of literals: *condition literals* and *task literals*. The condition literals encode the preconditions and effects of tasks or, in general, state variables for a process, while each task literal corresponds to a task that could occur in a process.

Capability rules have the following "*if*... *then*..." form: $r\colon l_1, \dots, l_n \Rightarrow l_{n+1}$, where r is a label that uniquely identifies the rule, and each literal l_i is drawn from the set of literals $\mathsf{Lit} = \mathsf{Prop} \cup \{\neg p | \, p \in \mathsf{Prop}\}$; Prop is a set of propositional atoms representing conditions c_i and tasks t_j. This form has the clear advantage that it immediately relates preconditions to the corresponding effect of performing the particular action. More specifically, we can identify the following three patterns: (i) $t \Rightarrow c$, where we can look at c as an effect of performing task t (the effect represented by c thus holds after the execution of task t); (ii) the pattern $c_1, \dots c_n \Rightarrow t$ indicates that $c_1, \dots c_n$ are preconditions for tasks t, and task t will be executed after the preconditions hold; (iii) $t_1, \dots, t_n \Rightarrow t$ specifies that the combination of tasks $t_1, \dots, t_n$ triggers task t, and that task t appears in the process, if $t_1, \dots, t_n$ appear in the process, before t. (In other words, this pattern describes relationships and dependencies among tasks in a process. In the rule given above, the meaning is that execution of tasks $t_1, \dots, t_n$ is required to trigger the execution of task t.)

The rules are then used to form (logical) derivations, where a *derivation* D, given a set of facts F represented as literals, is a sequence of literals $D(1), \dots, D(n)$, such that if $D(m + 1) = l$ then either $l \in F$ or there is a rule $r\colon l_1, \dots, l_k \Rightarrow l$ such that for all $l_i \in D[1..m]$, $i \leq k$, where $D[1..m]$ is the initial sequence of length m of D.

The rules presented above can be linked to the sequential, parallel, and alternative patterns typical of business process modelling techniques to those that can be found in a logical derivation. Indeed, assume tasks A and B concur to obtain the resources needed for task C to start its execution. This means that C may bring about its effects only when both A and B have finished, and that

A and B have no precedence order with respect to one another, that is they can be executed in parallel. From a logical perspective, all this information can simply be represented by a rule where the premises are literals A and B, and with S as conclusion. Accordingly, a derivation (sequence of rules) can encode a possible order in which the tasks are executed to achieve a particular business goal according to the constraints specified by the rules themselves.

Given a set of facts, we can generate a derivation where all applicable rules fire and their conclusions have been added to the derivation. This derivation contains all tasks that are executed given the set of facts (hence facts are the input for a process case). In addition, the derivation contains the literals corresponding to the conditions to trigger the execution of tasks or for activating obligations, the effects of the tasks, the obligations in force, and the expected goals. Notice that obligations and goals are neither actions nor tasks: they only purpose is to determine whether a process execution of the process is compliant and meet the organisation objectives (influencing thus the activities or tasks included in the process). Therefore, rules for goals and norms do not directly contribute to the structure of the process. Goals and obligations can thus be considered as *special kinds* of conditions. Consequently, if we "ignore" obligations, goals and condition literals from a derivation, then a derivation is a sequence of *only* those tasks satisfying the constraints defined by the rules. This is equivalent to a plan as defined in classical planning [11]. For these reasons, in the present paper, we concentrate only on the capability rules.

Notice that, while the set of tasks triggered by a case (set of facts) is unique, multiple derivations are possible. For instance, given the rule '$t_1, t_2 \Rightarrow t_3$', the order in which t_1 and t_2 are executed does not matter. Accordingly, both 't_1, t_2, t_3' and 't_2, t_1, t_3' are valid derivations (and, consequently, plans conforming to the specification given by the rule). This means that, given a case, we can generate a set of plans corresponding to it, which can be understood as alternative ways in which the process can be executed. Using the idea that a business process can be understood as a set of traces (where a trace is a sequence of tasks), we can establish a connection between a set of plans and a business process, where a process provides a concise (formal and graphical) representation of a set of plans, which are obtained from a single case and are combined by using constructions modelling AND-joins and AND-splits. Moreover, given a set of rules, it is possible to give as input different sets of facts, where each set of facts corresponds to a common set of instances for the process. For each case, a corresponding set of plans is created, where the mutually exclusive cases are subsequently merged, adopting XOR-split and XOR-join patterns.

3 Modal Defeasible Logic

The logic Governatori *et al.* [14,15] proposed to implement the intuition presented in Sect. 2 is a modal extension of the well-known skeptical formalism of Defeasible Logic (DL), first introduced in a seminal work by Nute [27]. Specifically, the logic deploys the non-monotonic mechanism of DL to capture:

(i) which actions an agent (enterprise) is capable to perform in a given organisational environment by using a *belief* modality (in other terms, *beliefs* describe both what holds true in the environment as well as which actions the agent is able to perform), (ii) which norms the agent is subject to (by using the *obligation* modality), and (iii) which goals the agent might commit to and which are actually attainable (by using the *outcome* modality).

A modal defeasible theory consists of five different kinds of knowledge: facts, strict rules, defeasible rules, defeaters, and a superiority relation. The set of facts denotes simple pieces of information that are considered always to be true. Rules are distinguished both on their *type* (strict, defeasible and defeaters) and on their *modality.* Such a modality represents which kind of conclusion the rule may lead to. Rules are of three modalities: belief rules, obligation rules, and outcome rules. Belief rules are meant to relate the factual knowledge of an enterprise, and are composed by: a set of actions an organisation can do, the *preconditions* under which tasks can be executed, and the effects derived by the execution of such tasks (also called *postconditions*). Specifically, *belief rules* describe the logical relationship between preconditions and tasks, tasks and their effects, relationships between tasks, relationships between states. *Obligation rules* determine when and which obligations are in force, while outcome rules establish the objectives of an organisation depending on the particular context. *Outcome rules* take inspiration by the main stream of the BDI (Belief-Desire-Intention) literature describing an agent's mental attitudes. Notions of desire, goal, intention, and social intention are taken into account in order to describe various degrees of the agent's commitment towards its objectives.

A *rule* is an expression $r : A(r) \hookrightarrow_\Box C(r)$ and consists of: (i) a unique name r, (ii) the *antecedent* $A(r)$ that which is a finite set of (modal) literals, (iii) an arrow $\hookrightarrow \in \{\rightarrow, \Rightarrow, \leadsto\}$ denoting, respectively, a strict rule, a defeasible rule and a defeater, (iv) a modality $\Box \in \{BEL, OBL, OUT\}$ that which denotes the mode the consequent literal shall take, i.e. BEL for beliefs, OBL for obligations and OUT for outcomes (outcomes are themselves distinguished among desires, goals, intentions, and social intentions, those being intentions compliant with the norms (it is out of the scope of the present synopsis to go in further details by describing differences among such mental attitudes), (v) its *consequent* (or *head*) $C(r)$ that which is a single literal, and (vi) the superiority relation $>$ that which is a binary relation indicating the relative strength of two (conflicting) rules. A *strict rule* is a rule in which whenever the premises hold (e.g., facts), so does the conclusion. On the other hand, a *defeasible rule* is a rule that can be defeated by contrary evidence (typically bird fly, but it is not the case for penguins). *Defeaters* are rules that cannot be used to draw any conclusion. Their only use is to prevent some conclusions, i.e., to defeat defeasible rules by producing evidence for the contrary. Lastly, the superiority relation $>$ among rules is used to define when one rule overrides the (opposite) conclusion of another one. The infix notation $r > s$ means that $(r, s) \in >$.

At the heart of the reasoning mechanism of the logic is the notion of derivation. Intuitively a *derivation* (or *proof*) is a sequence of literals where every element (a conclusion) is either one of the facts, or it has been obtained by previous steps by applying some rules. A *conclusion* of a defeasible theory D is a tagged literal and can have one of the following forms:

- $+\Delta_{\Box}q$, which means that q is definitely provable in D with mode $\Box$, i.e., there is a definite proof for q, that is a proof using facts and strict rules only;
- $-\Delta_{\Box}q$, which means that q is definitely not provable in D with mode $\Box$ (i.e., a definite proof for q does not exist);
- $+\partial_{\Box}q$, which means that q is defeasibly provable in D with mode $\Box$;
- $-\partial_{\Box}q$, which means that q is not defeasibly provable, or refuted in D with mode $\Box$.

Formally, given a defeasible theory D, a proof P of length n in D is a finite sequence $P(1), \ldots, P(n)$ of tagged formulas of the type $+\Delta_{\Box}q$, $-\Delta_{\Box}q$, $+\partial_{\Box}q$ and $-\partial_{\Box}q$.

The definition of Δ describes just forward chaining of strict rules. Literal q is definitely provable if either is a fact, or there is a strict rule for q, whose antecedents have all been previously, definitely proved. On the other hand, literal q is defeasibly provable $(+\partial_{\Box}q)$ if q is already definitely provable, or we argue using the defeasible part of the theory. In this last case, there must exist an applicable strict or defeasible rule for q, while every attack is either discarded, or defeated by a stronger rule through $>$. (A rule is merely *applicable* whenever each literal in the set of antecedents has already been proved, while a rule is *discarded* when at least one of the premises has been previously disproved.)

The sets of positive and negative conclusions form, respectively, the *positive* and *negative extensions*. For reasons we are going to explain in the rest of the paper, we shall restrict our attention to the positive extension only.

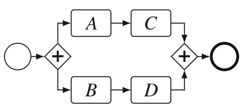
Let us explain how the derivation mechanism works by considering the following two examples. The first one proposes a rather simple theory; aim is to link derivations to traces. We use second example to show how strict and defeasible proof tags are obtained, as well as the mechanism underlying the team defeat.

Let $D = (\{a, b\}, R, \emptyset)$ be a defeasible theory such that

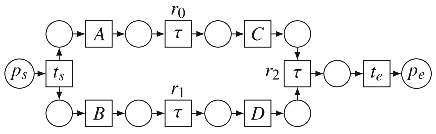
$$\begin{aligned} R = \{ & r_0 : a \Rightarrow_{BEL} c \\ & r_1 : b \Rightarrow_{BEL} d \\ & r_2 : c, d \Rightarrow_{BEL} e \}. \end{aligned}$$

It is trivial to notice that all the rules are applicable and actually fire producing the following positive extension $E^+(D) = \{+\Delta_{BEL}a, +\Delta_{BEL}b, +\partial_{BEL}a, +\partial_{BEL}b, +\partial_{BEL}c, +\partial_{BEL}d, +\partial_{BEL}e\}$ – recall that $+\Delta a$ implies $+\partial a$. Here, we have six derivations:

$$(1) + \Delta_{BEL}a, +\Delta_{BEL}b, +\partial_{BEL}c, +\partial_{BEL}d, +\partial_{BEL}e$$
$$(2) + \Delta_{BEL}a, +\Delta_{BEL}b, +\partial_{BEL}d, +\partial_{BEL}c, +\partial_{BEL}e$$



(a) BPMN.



(b) Petri net.

Fig. 1. Two processes representing the traces of theory D.

$$(3) + \Delta_{BEL}a, +\partial_{BEL}c, +\Delta_{BEL}b, +\partial_{BEL}d, +\partial_{BEL}e$$
$$(4) + \Delta_{BEL}b, +\Delta_{BEL}a, +\partial_{BEL}c, +\partial_{BEL}d, +\partial_{BEL}e$$
$$(5) + \Delta_{BEL}b, +\Delta_{BEL}a, +\partial_{BEL}d, +\partial_{BEL}c, +\partial_{BEL}e$$
$$(6) + \Delta_{BEL}b, +\partial_{BEL}d, +\Delta_{BEL}a, +\partial_{BEL}c, +\partial_{BEL}e.$$

Some considerations. a and b have no precedence order between each other. The same happens for the tuples: (c, d), (a, d), and (b, c). On the contrary, a always precedes c, b always precedes d, and so do c and d for e. It is straightforward to see that such derivations can be visualised as proposed in Fig. 1, where Fig. 1(a) shows a BPM notation whilst Fig. 1(b) shows a Petri net.

Now consider the next example and let $D = (F = \{c_1, c_2, c_3, c_4, c_5, c_7, c_8\}, R, \succ= \{(r_6, r_9), (r_7, r_8)\})$ be a defeasible theory such that

$$\begin{aligned}
R = \{r_0 &: c_1, c_2 \rightarrow_{BEL} t_1; & r_1 &: c_3 \Rightarrow_{BEL} t_2;\\
r_2 &: t_1, t_2 \Rightarrow_{BEL} c_6; & r_3 &: t_2 \Rightarrow_{OBL} o;\\
r_4 &: c_6 \Rightarrow_{BEL} t_4; & r_5 &: t_4 \Rightarrow_{BEL} o;\\
r_6 &: c_4 \Rightarrow_{BEL} t_3; & r_7 &: c_5, OBLo \Rightarrow_{BEL} t_3;\\
r_8 &: c_7 \Rightarrow_{BEL} {\sim}t_3; & r_9 &: c_8 \Rightarrow_{BEL} {\sim}t_3;\\
r_{10} &: \Rightarrow_{OUT} out; & r_{11} &: t_3 \Rightarrow_{BEL} out\}.
\end{aligned}$$

We denoted tasks with t_i, conditions with c_j, the only obligation with o, and the final outcome with *out*. In detail, r_0 strictly proves $+\Delta_{BEL}t_1$, while r_1 defeasibly proves $+\partial_{BEL}t_2$. The combination of these two tasks gives (defeasibly proves) condition c_6, but the execution of t_2 also triggers obligation o (by proving $+\partial_{OBL}o$). Such an obligation is complied with by the execution of task t_4 through the sequential derivation of $+\partial_{BEL}t_4$ by r_4 and $+\partial_{BEL}o$ by r_5. Now, rules r_6 and r_7 form a (winning) team defeater to prove t_3: the superiorities among $r_6 \succ r_9$ and $r_7 \succ r_8$, let r_8 and r_9 to be defeated. We hence have two ways to obtain t_3. The only outcome is derived by r_{10}, which is obtained through r_{11}. The positive extension is $E^+(D) = \{+\Delta_{BEL}c_1, \ldots, +\Delta_{BEL}c_6, +\partial_{BEL}c_1, \ldots, +\partial_{BEL}c_6, +\Delta_{BEL}t_1, +\partial_{BEL}t_1, +\partial_{BEL}t_2, +\partial_{OBL}o, +\partial_{BEL}c_7, +\partial_{BEL}t_3, +\partial_{BEL}t_4, +\partial_{BEL}o, +\partial_{OUT}out, +\partial_{BEL}out\}$.

4 Visualisation Methods for Compliant Processes

We introduced a modal logic describing (1) what are the sequences of actions (in terms of literals derivations) an organisation is able/allowed to perform in a given setting/situation (2) to achieve a set of goals (3) while remaining norm compliant with a regulative system. The next step is to define the methods to represent/visualise such derivations (traces) in a compact, explicative manner. We shall report two different approaches. The former, proposed by Olivieri *et al.* in [28,29], adopts a backwards approach and ends up in visualising the process through the BPM Notation. The latter, recently proposed by Ghanbari *et al.* in [12], is otherwise based on the Petri net modelling language.

The algorithms proposed in [14,15] take as inputs (i) a modal logics and (ii) (factual) literals describing a specific situation. The output is the *positive extension* expressing which literals actually hold in that particular setting. This reflects which norms are active and which tasks can be performed by the organisation. Such literals and the rules where such literals appear in as antecedents or as conclusions are used by both methods to visualise the final norm and goal compliant business processes.

4.1 BPMN and the Backwards Approach

We hereafter analyse the approach proposed by Olivieri *et al.* [14,15]. The logic described in Sect. 3 is expressive enough to be able to describe most-preferred to least-preferred objectives. (Such objectives lie within the agent BDI paradigm and are represented as reparative chains, where an *outcome chain* like '$a \odot b$' characterises the idea that a is the most preferred outcome, but when a is not attainable, then b becomes the new outcome the agent strives for. (It is out of the scope of the present paper to go further in detail on outcome chains and their implications on the various degrees of the agent's commitment towards its goals.)

The algorithms of [14,15] work in a backwards manner. They start by considering the *end literals* of the theory in the positive extension (the proved ones), those representing the attainable outcomes. Exclusive choice patterns are created among the outcomes from the same chain. The algorithms then navigate backwards the derivation tree, rule by rule, until the facts of the theory are met. Accordingly, only those rules with both conclusion and set of antecedents in the positive extension are considered. Every time the algorithm considers a (non-already visited) literal, a new node is created in the graph. Given a rule, an edge is created between the antecedent of such a rule and its conclusion. If more than one literal is present in the set of antecedents, an AND-join node is created in between each literal in the set of antecedents and the conclusion. Finally, if more than one rule contributes in proving a given literal, an OR-join structure is made. The final steps of the algorithms consist in giving "more structure" to the graph: (i) nodes representing condition and obligation literals are removed and substituted by labels on edges, (ii) literals *co-occurrences* are identified, and lastly (iii) complex synthesis operations on the graph are performed to create OR-split and AND-join patterns. The algorithms are proved to be sound, complete, and to work in polynomial time.

4.2 The Petri Net Approach

In [12], a formal method to visualise and operationalise business rules in the form of a Petri net is presented. (The reader in need is referred to Appendix A for a brief excerpt on the Petri net formalism.) Figure 2 provides an overview of the steps required in [12] to transform a set of rules into a Petri net, representing the allowed behaviour of the business process according to those rules.

The process to be obtained is imperative and exclusively contains possible execution paths. That is, it defines what *can* or *should* be executed, instead of what *must not* be executed (as is normally the case for declarative process specifications). Consequently, the rules are first pre-processed to remove those rules that do not directly define possible executions of the process. As such, all rules with literals that have not been proved can be removed, as these rules cannot fire and have, therefore, no effect on the resulting process. Additionally, negative tasks represent the absence of a task and can thus be removed from the remaining rules (e.g., $A, \neg B \Rightarrow C$ would be $A \Rightarrow C$).

The rules are generally grouped in different input cases, each representing a specific "scenario" or process instance. For each case, a partial Petri net is obtained, representing the process according to the rules activated (i.e., rules with antecedents and conclusion proved) for that case. That is, the traces of each partial Petri net contain exactly all possible derivations from the rules of its corresponding case. Each literal is represented by a transition in the Petri net, whereas each rule is represented by a τ transition. As a result, each partial Petri net essentially contains a sequence of transitions representing subsequent activities and rules. Multiple rules with an identical antecedent and different consequence result in concurrent branches in the partial Petri net. For instance, rules $A \Rightarrow B$ and $A \Rightarrow C$, would introduce two concurrent paths (with B and C, respectively) after A, whereas $B, C \Rightarrow D$ would merge both branches into a single path with D. Each partial Petri net consists of sequences and/or concurrent branches, as choices are represented by the different cases.

Subsequently, the partial Petri nets are merged into a consolidated Petri net, representing the full process such that it does not contain any duplicate transition labels (i.e., each activity is represented only once). Different paths following a

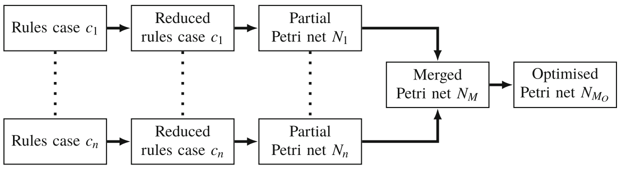


Fig. 2. Method overview (derived from [12]).

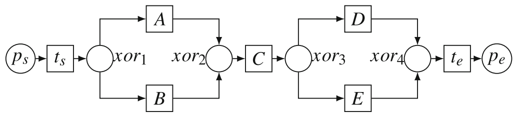


Fig. 3. Consecutive XORs without dependencies.

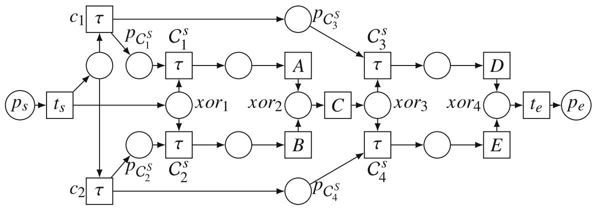


Fig. 4. Consecutive XORs preserving dependencies.

mutual transition between different partial Petri nets result in exclusive paths in the merged Petri net.

However, subsequent exclusive choices may not necessarily be independent. That is, the allowed path at a certain XOR-split may be determined by the preceding path from the previous XOR-split. Consider, for instance, two cases: *Case 1* $= \{r_1 :\Rightarrow A, r_2 : A \Rightarrow C, r_3 : C \Rightarrow D\}$ and *Case 2* $= \{r_4 :\Rightarrow B, r_5 : B \Rightarrow C, r_6 : C \Rightarrow E\}$. The (simplified) resulting merged Petri net would then look as shown in Fig. 3. It is easy to see that the merged Petri net would allow two more traces that are not allowed in the original derivations as specified by Cases 1 and 2 (i.e., $\langle A, C, E\rangle$ and $\langle B, C, D\rangle$ are not allowed).

This is resolved by adding τ transitions representing the underlying cases that preserve dependencies of subsequent exclusive branches, without the necessity of adding conditions. This is shown graphically in Fig. 4, which represents the same process as depicted in Fig. 3 while maintaining consecutive dependencies as specified in the rules. As such, after A, E is not possible, while D is not possible in a trace with B.

Finally, unnecessary τ transitions (i.e., transitions whose removal does not alter the possible visible traces of the net) can be removed by using the reduction techniques as described in [26]. The resulting Petri net is an exact imperative representation of the behaviour as allowed by the input rules.

5 A Brief Analysis of the Two Approaches

5.1 Pros and Cons of the BPMN Approach

Pros:

The modal logics presented before is rich of information regarding all the possible actions and norms; moreover, the calculus of the extension is not only computationally efficient but also gives precise information concerning what is *actually* attainable in every specific situation, and which norms are actually in force.

– Methodologies of [28,29] fully exploit such information: the backwards approach guarantees that no information is lost during the computation. Therefore, the final graph shows which *condition literals are used* and where, as well as which norms impact on the process and, again, exactly where (i.e., which tasks are influenced).
– We recall that the starting point of the synthesis algorithms was to consider which the attainable outcomes were: those were embedded in exclusive choice structures. This gives an immediate feedback to the front-end user about which the alternative (the most to the least preferred) outcomes are in a given setting.
– The merging process of different graphs (each of which representing different initial settings) gives as input a single structured process graph where the various alternatives are represented through XOR structures. Many XOR variants were considered, depending on whether (i) there exists a preferred XOR branch to be executed, (ii) none of the branches involved are to be preferred to the others, and (iii) a branch that does not actually involve the execution of any task and it is used to skip the run to the end of the XOR-join (which is typically considered as the standard course of action).

Cons:

– The merging procedure is not fully operational: the algorithms can only handle the creation of the XOR structures but is not even close to the deepness reached by methods proposed in [12].
– A proper proof of completeness is missing.
– The OR-join gates created when more rules concur in achieving a node cannot properly handle the *resource consumption*.

5.2 Pros and Cons of the Petri Net Approach

Pros:

– This approach has proven to be fast and has shown to outperform state-of-the-art approaches.
– The resulting Petri net only allows behaviour that is allowed by the rules and is, as such, guaranteed to be compliant to those rules. More specifically, a condition-free representation of subsequent exclusive branches is created to maintain their dependencies without duplicating activities. When required, these dependencies can be removed automatically to generalise the behaviour of the Petri net. Existing approaches, however, can only obtain a generalised Petri net and are, therefore, in many cases not fully compliant with the input rules.
– Full proofs of soundness and completeness were given.

Cons:

– The approach does not consider processes with loops (for the same reasons of the other approach).

– Subsequent dependencies are ensured and enforced by creating a specific coordinating transition for each input case. Naturally, this may complicate the resulting Petri net in scenarios with many input cases. However, the number of cases can be significantly reduced by merging, as together they may represent full behaviour. Merging such cases is a direction for future work.

6 Related Work

Some other approaches attempted to create compliant business process by design.

The defeasible modal logic we presented departs from the standard BDI architecture and agent programming languages implementing it (e.g., 3APL-2APL [6,7], Jason [3]), and extensions with norms in several respects (e.g., BOID by [4], while we refer the reader to [1] for an overview). While in the above mentioned approaches the agent has to select (partially) predefined plans from a plan library, we propose that the agent generates a business process on the fly (corresponding to a set of plans) to meet the objectives without violating the norms it is subject to.

Alechina *et al.* [2] present a BDI-based agent programming language based on 2APL for *norm-aware agents*; a norm-aware agent can deliberate on its goals, norms, and sanctions before deciding which plan to select and execute. A major issue of this work is that if a goal triggers two (or more) sanctions, each of which is lower in rank than the achievement of that goal, the agent will try to achieve that particular goal even if the sum of the two sanctions is higher in rank.

Automated planning is a technique to organise actions with the aim of achieving some pre-specified goal starting from the current state of the system [11]. Each action features a set of preconditions, that must be satisfied prior to its execution, and a set of effects, that specify the state change resulting from its execution. There are frameworks to generate plans (e.g., KPG [22] and Golog [10]), but these are typically based on classical AI planning and do not consider norms. In addition, many automated planning approaches in the business process domain focus on runtime adaptation of pre-specified processes, concerning runtime repair instead of design time process generation based on rules (see [23,36]). Automated planning techniques require a goal to be specified along with an initial state. However, in case of multiple initial states, multiple possible plans need to be generated, that must be merged in order to represent a full business process [21]. As such, providing a business process model that supports *all* possible traces as specified by the rules remains a challenge.

DECLARE provides an approach for declarative specifications of business processes by means of constraints [30] and graphical representations to visualise the constraints and the activities in the model [40]. However, the graphical representation shows the exact constraints and does not provide an actual process model imposed by the rules. In [32], Prescher *et al.* convert a DECLARE model to a behaviourally equivalent Petri net. However, this approach leads to multiple transitions representing the same activity and, therefore, an highly complicated model. Our approach, on the other hand, provides a model without additional duplication of transitions. In [8], Giacomo *et al.* developed an extension on top of BPMN, BPMN-D, which supports declarative process modelling. It allows to transform DECLARE models into readable BPMN-D models. This approach, however, does not support concurrency. Additionally, DECLARE is based on Linear Temporal Logic, which is not able to represent certain complex norms [13] and cannot, as such, be used for this purpose.

Sardina *et al.* [35] provide an account of goals in the view of declarative aspects by integrating BDI failure mechanisms with Hierarchical Task Network (HTN) planning techniques. HTN planning is notoriously undecidable even if no variables are allowed, or PSPACE-hard if restrictions are given. The main feature of their CAN$^{\mathcal{A}}$ is its detailed operational semantics where, if a plan fails, alternative plans for achieving the goal are tried. Compared to theirs, the two approaches presented in this paper have the advantage that they generate all possible plans at design time.

7 Future Work

Business Process Compliance is an important field of study given the importance for enterprises to have business processes which have to be, at the same time, efficient and compliant with the normative system. Scholars of the fiels have studied, and proposed, different formalisms to describe workflows, business processes, and norms. In the present work, we described two promising approaches which lie in the school of modelling business processes by declarative specifications. This school of thoughts differs from the "more stiffed" family of imperative approaches since it gives knowledge about the relationships among tasks, but most of all because it allows us to represent in the same framework business and normative specifications.

The modal logics [14,15] described in Sect. 3 is a powerful tool exactly for this reason. Still, some drawbacks need to be addressed in future lines of research: (i) loops are not considered, and (ii) resources consumption.

A Petri Nets

Petri nets (PN) are a popular modelling language used to formalise business processes [37]. Petri nets are mathematical models for the description of distributed systems [31]. Petri nets are directed bi-graphs with nodes consisting of *places* and *transitions.* Transitions within Petri nets represent events, while places represent conditions. *Arcs* form directed edges between place-transition pairs. Places may contain *tokens.* A distribution of tokens over the places is called a *marking.* A transition is enabled and can "fire" when all its input places contain at least one token. When a transition fires, one token is removed from each input place and one token is put into each output place. A Petri net is defined formally as follows [31]:

Definition 1 (Petri net). *A tuple* (P, T, A, λ) *is a* labeled Petri net*, where:*

- P *is a set of* places
- T *is a set of* transitions*, such that* $P \cap T = \emptyset$
- $A \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs*
- $\lambda : P \cup T \rightarrow \mathcal{L}$ *is a labelling function.*

The Petri net state, often referred to as the net marking*,* $M : P \rightarrow \mathbb{N}_0$ *is a function that associates a place* $p \in P$ *with a natural number (viz., place tokens). A* marked net $N = (P, T, A, \lambda, M_0)$ *is a Petri net* (P, T, A, λ) *together with an* initial marking M_0.

Places and transitions are referred to as *nodes*. The *preset* of a node is denoted by $\bullet y = \{x \in P \cup T \mid (x, y) \in A\}$, and the *postset* of a node is denoted by $y\bullet = \{z \in P \cup T \mid (y, z) \in A\}$.

If $\forall p \in \bullet t : M(p) > 0$, t is said to be *enabled*. The *firing* of t, denoted by $M \xrightarrow{t} M'$, leads to a new marking M', with $M'(p) = M(p) - 1$ if $p \in \bullet t \setminus t\bullet$, $M'(p) = M(p) + 1$ if $p \in t\bullet \setminus \bullet t$, and $M'(p) = M(p)$ otherwise. The marking M_n is said to be reachable from M if there exists a sequence of transition firings $\sigma = t_1 t_2 \ldots t_n$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$.

A *trace* is a sequence $\lambda(t_1), \lambda(t_2), \ldots$ such that $\sigma = t_1, t_2, \ldots$ is a sequence of firing transitions. However, certain control-flow behaviour (like exclusive parallel branches) requires additional transitions that do not correspond to a task literal. These transitions are commonly referred to as silent or τ transitions [9]. For understandability purposes, we will add a label for each τ transition as well throughout the paper. As such, the set of transition labels $\mathcal{L}$ comprises both labels corresponding to task literals and labels corresponding to τ transitions. A *visible trace* is a trace where all τ transitions have been removed (maintaining the order of the transitions representing task literals). For the remainder of this work, we shall refer to visible traces as traces.

References

1. Alechina, N., Bassiliades, N., Dastani, M., Vos, M.D., Logan, B., Mera, S., Morris-Martin, A., Schapachnik, F.: Computational models for normative multi-agent systems, pp. 71–92
2. Alechina, N., Dastani, M., Logan, B.: Programming norm-aware agents, pp. 1057–1064
3. Bordini, R.H., Hübner, J.F.: BDI agent programming in agentspeak using *Jason*. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 143–164. Springer, Heidelberg (2006). https://doi.org/10.1007/11750734_9. (tutorial paper)
4. Broersen, J., Dastani, M., Hulstijn, J., van der Torre, L.W.N.: Goal generation in the BOID architecture. Cogn. Sci. Q. **2**(3–4), 428–447 (2002)
5. Chesani, F., Mello, P., Montali, M., Riguzzi, F., Sebastianis, M., Storari, S.: Checking compliance of execution traces to business rules. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008. LNBIP, vol. 17, pp. 134–145. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00328-8_13

6. Dastani, M.: 2APL: a practical agent programming language. Auton. Agent. Multi-Agent Syst. **16**(3), 214–248 (2008)
7. Dastani, M., van Riemsdijk, M.B., Meyer, J.J.C.: Programming multi-agent systems in 3APL. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah, S.A. (eds.) Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations (International Book Series), vol. 15, pp. 39–67. Springer, Boston (2005). https://doi.org/10.1007/0-387-26350-0_2
8. De Giacomo, G., Dumas, M., Maggi, F.M., Montali, M.: Declarative process modeling in BPMN. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 84–100. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_6
9. Dijkman, R., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
10. Gabaldon, A.: Making golog norm compliant. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) CLIMA 2011. LNCS (LNAI), vol. 6814, pp. 275–292. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22359-4_19
11. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann, Burlington (2004)
12. Ghooshchi, N.G., van Beest, N.R.T.P., Governatori, G., Olivieri, F., Sattar, A.: Visualisation of compliant declarative business processes. In: Proceedings of EDOC. IEEE (2017, forthcoming)
13. Governatori, G., Hashmi, M.: No time for compliance. In: Proceedings of the International Conference on Enterprise Distibuted Object Computing, pp. 9–18. IEEE (2015)
14. Governatori, G., Olivieri, F., Rotolo, A., Scannapieco, S., Cristani, M.: Picking up the best goal: an analytical study in defeasible logic. In: Morgenstern, L., Stefaneas, P., Lévy, F., Wyner, A., Paschke, A. (eds.) RuleML 2013. LNCS, vol. 8035, pp. 99–113. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39617-5_12
15. Governatori, G., Olivieri, F., Scannapieco, S., Rotolo, A., Cristani, M.: The rationale behind the concept of goal. TPLP **16**(3), 296–324 (2016)
16. Governatori, G., Padmanabhan, V., Rotolo, A., Sattar, A.: A defeasible logic for modelling policy-based intentions and motivational attitudes. Log. J. IGPL **17**(3), 227–265 (2009)
17. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: Link, S., Ghose, A. (eds.) APCCM. CRPIT, vol. 110, pp. 3–12. Australian Computer Society (2010)
18. 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). https://doi.org/10.1007/978-3-642-16289-3_17
19. Governatori, G., Sadiq, S.: The journey to business process compliance. In: Handbook of Research on BPM, pp. 426–454 (2008)
20. Hagerty, J., Hackbush, J., Gaughan, D., Jacobson, S.: The governance, risk management, and compliance spending report, 2008–2009: inside the $32B GRC market. AMR Research (2008)
21. Heinrich, B., Schön, D.: Automated planning of process models: the construction of simple merges. In: 24th European Conference on Information Systems (ECIS) (2016)

22. Kakas, A.C., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: The KGP model of agency. In: ECAI, pp. 33–37. IOS Press (2004)
23. Marrella, A., Mecella, M.: Continuous planning for solving business process adaptivity. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS/EMMSAD -2011. LNBIP, vol. 81, pp. 118–132. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21759-3_9
24. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100**(1), 1–40 (1992)
25. Morgan, T.: Business Rules And Information Systems: Aligning IT With Business Goals. Addison-Wesley Professional, Reading (2002)
26. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
27. Nute, D.: Defeasible logic. In: Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 3. Oxford University Press (1987)
28. Olivieri, F., Cristani, M., Governatori, G.: Compliant business processes with exclusive choices from agent specification. In: Chen, Q., Torroni, P., Villata, S., Hsu, J., Omicini, A. (eds.) PRIMA 2015. LNCS (LNAI), vol. 9387, pp. 603–612. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25524-8_43
29. Olivieri, F., Governatori, G., Scannapieco, S., Cristani, M.: Compliant business process design by declarative specifications. In: Boella, G., Elkind, E., Savarimuthu, B.T.R., Dignum, F., Purvis, M.K. (eds.) PRIMA 2013. LNCS (LNAI), vol. 8291, pp. 213–228. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-44927-7_15
30. Pesic, M., Schonenberg, H., Van der Aalst, W.M.: Declare: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2007, pp. 287–300. IEEE Computer Society (2007)
31. Petri, C.A.: Communication with automata, Ph.D. thesis. Universität Hamburg (1966)
32. Prescher, J., Di Ciccio, C., Mendling, J.: From declarative processes to imperative models. SIMPDA **14**, 162–173 (2014)
33. Russell, N.C., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Designing a workflow system using coloured petri nets. In: Jensen, K., Billington, J., Koutny, M. (eds.) Transactions on Petri Nets and Other Models of Concurrency III. LNCS, vol. 5800, pp. 1–24. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04856-2_1
34. Sadiq, S., Governatori, G.: Managing regulatory compliance in business processes. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 2. IHIS, pp. 265–288. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-642-45103-4_11
35. Sardiña, S., Padgham, L.: A BDI agent programming language with failure handling, declarative goals, and planning. Auton. Agent. Multi-Agent Syst. **23**(1), 18–70 (2011)
36. van Beest, N.R.T.P., Kaldeli, E., Bulanov, P., Wortmann, J.C., Lazovik, A.: Automated runtime repair of business processes. Inf. Syst. **39**, 45–79 (2014)
37. van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)
38. van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. Inf. Softw. Technol. **41**(10), 639–650 (1999)

39. van der Aalst, W.M.P., Pesic, M.: Decserflow: towards a truly declarative service flow language. In: Leymann, F., Reisig, W., Thatte, S.R., van der Aalst, W.M.P. (eds.) The Role of Business Processes in Service Oriented Architectures, Dagstuhl Seminar Proceedings, vol. 06291. Schloss Dagstuhl, Germany (2006)
40. Westergaard, M., Maggi, F.M.: Declare: a tool suite for declarative workflow modeling and enactment. BPM (Demos) **820**, 1–5 (2011)