Business Processes Verification with Temporal Answer Set Programming *

L. Giordano[1], A. Martelli[2], M. Spiotta[1], and D. Theseider Dupré[1]

[1] Dipartimento di Informatica, Università del Piemonte Orientale
[2] Dipartimento di Informatica, Università di Torino

Abstract. The paper provides a framework for the specification and verification of business processes, based on a temporal extension of answer set programming (ASP). The framework allows to capture fluent annotations as well as data awareness in a uniform way. It allows for a declarative specification of business process but also for a direct encoding of processes specified in conventional workflow languages. Verification of temporal properties of a business process, including verification of compliance to business rules, can be performed by LTL bounded model checking techniques.

1 Introduction

The verification of business process compliance to business rules and regulations has gained a lot of interest in recent years and it has led to the development to a process annotation approach [12, 18, 33, 23], where a business processes is enriched with information relevant for compliance verification, to capture the semantics of atomic tasks execution through preconditions and effects. The treatment of data in business process verification, on the other hand, has attracted growing interest in the last decade, with the definition of *artifact-centric* and *data-centric* process models [27, 5, 9].

In this paper we combine the two perspectives and propose a framework for the specification and verification of business processes which allows to model both annotations and data properties by specifying atomic tasks in a uniform way. The approach is well suited for a declarative specification of the business process, which has been advocated by many authors in the literature [32, 30, 25]. Following [7], the specification of annotation can be done in an action theory by defining the effects and preconditions of atomic tasks. The same approach allows to capture data properties, by modelling data acquisition tasks as actions which nondeterministically assign values to variables (data objects) on given domains, under the restriction that domains are finite.

The use of directional rules for modeling business rules as well as to capture the conditional structure of norms is widely used in the literature [18]. In our approach, besides the specification of action preconditions and direct effects, *causal rules* in an action domain allow to capture dependencies among fluents

* This work has been partially supported by Regione Piemonte, Project ICT4LAW.

(propositions whose truth is affected by actions) and *fluent changes*, as well as dependencies between process data and fluents. Our claim is that both static and dynamic causal laws are useful for the specification of business process annotations and their use allows unintended conclusions to be avoided. Observe that, once the data perspective is included, causal laws can include both conditions on data and annotations. For instance, the rule $age \geq 18 \Rightarrow ofAge$ may establish a link between the business process, whose execution assigns values to the variable *age*, and the compliance rules dealing with persons "of age".

The approach we propose is based on Answer Set Programming (ASP) [11] and, more precisely, on the temporal extension of ASP in [16], combining ASP with the temporal logic DLTL [22], an extension of LTL in which the temporal operators are enriched with program expressions. The action language in [16] allows general DLTL constraints to be included in action domains, which can be profitably used for a declarative specification of the business process advocated in the literature [32, 30, 25]. In addition, the proposed approach also allows for a direct encoding of processes specified in workflow languages, and it can be used in combination with state of the art workflow management systems.

The paper considers several verification tasks including the verification of business process compliance to business rules. Verification is performed through Bounded Model Checking [6] techniques and exploits the approach in [16] for DLTL bounded model checking in ASP, which extends the approach for Bounded LTL Model Checking with Stable Models in [21].

2 A Temporal Answer Set Programming language

In this section we recall the temporal ASP language introduced in [16]. The language is based on a temporal extension of Answer Set Programming (ASP) which combines ASP with the temporal logic DLTL [22], an extension of LTL in which temporal operators are enriched with program expressions. In particular, in DLTL the next state modality can be indexed by actions, and the until operator $\mathcal{U}^\pi$ can be indexed by a program π which, as in PDL, can be any regular expression built from atomic actions using sequence (;), nondeterministic choice (+) and finite iteration ($*$). Satisfiability and validity for DLTL are PSPACE-complete problems [22].

Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite non-empty alphabet of actions. From the until operator, the derived modalities $\langle\pi\rangle$, $[\pi]$, $\bigcirc$ (next), $\mathcal{U}$, $\Diamond$ and $\Box$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$, $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$, $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Informally, a formula $[\pi]\alpha$ is true in a world w of a linear temporal model if α holds in all the worlds of the model which are reachable from w through any execution of the program π. A formula $\langle\pi\rangle\alpha$ is true in a world w of a linear temporal model if there exists a world of the model reachable from w through an execution of the program π, in which α holds.

A *domain description* D is a pair $(\Pi, \mathcal{C})$, where Π is a set of laws describing the effects and executability preconditions of actions (as described below), and $\mathcal{C}$

is a set of *temporal constraints*, i.e., general DLTL formulas. Atomic propositions describing the state of the domain are called *fluents*. Actions may have direct effects, described by action laws, and indirect effects, described by causal laws capturing the causal dependencies among fluents.

Let $\mathcal{L}$ be a first-order language which includes a finite number of constants and variables, but no function symbol. Let $\mathcal{P}$ be the set of predicate symbols, Var the set of variables and C the set of constant symbols. We call *fluents* atomic literals of the form $p(t_1, \ldots, t_n)$, where, for each i, $t_i \in Var \cup C$. A *simple fluent literal* l is an atomic literal $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$. We denote by Lit_S the set of all simple fluent literals, and we assume that the fluent $\bot$ representing the inconsistency is included in Lit_S. A *temporal fluent literal* has the form $[a]l$ or $\bigcirc l$, where $l \in Lit_S$ and a is an action name (an atomic proposition, possibly containing variables). Given a (simple or temporal) fluent literal l, *not* l represents the default negation of l. A (simple or temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*. The laws are formulated as rules of a temporally extended logic programming language having the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{1}$$

where the l_i's are simple or temporal fluent literals. As usual in ASP, rules with variables are a shorthand for the set of their ground instances; and we let Σ be the set of ground instances of atomic actions in the domain description.

In the following we call a *state* a set of ground fluent literals. A state is said to be *consistent* if it is not the case that both f and $\neg f$ belong to the state, or that $\bot$ belongs to the state. The execution of an action in a state may possibly change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state. We assume that a law as (1) can be applied in all states while, when prefixed with the **Init**, it only applies to the initial state.

Action laws, causal laws, precondition laws, persistency laws, initial state laws, etc., which are normally used in action theories, can all be defined as instances of (1). *Action laws* describe the effects of atomic tasks. The meaning of an action law $[a]l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$, (where $l_0 \in Lit_S$ and $l_1, \ldots, l_n$ are either simple fluent literals of temporal fluent literals of the form $[a]l$) is that executing action a in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold makes the effect l_0 to hold (in the state after the action).

Precondition laws allow the specification of executability conditions for atomic tasks; they are a special case of action laws with $\bot$ as effect, i.e., they have the form: $[a]\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ meaning that a cannot be executed (has an inconsistent effect) in case $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold.

Causal laws define causal dependencies among propositions, which are used to derive indirect effect of actions, called *ramifications* in the literature of reasoning about actions where it is well known that causal dependencies among propositions are not suitably represented by material implication in classical logic. *Static causal laws* have the form: $l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ where the l_i's are fluent literals. Their meaning is: if $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold in a state, then l_0 is caused to hold in that state. *Dynamic causal laws* have the

form: $\bigcirc l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n$ where l_0 is a fluent literal and the t_i's are either fluent literals or temporal fluent literals of the form $\bigcirc l_i$ (meaning that the fluent literal l_i holds in the next state). Their meaning is: if $t_1, \ldots, t_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold, then l_0 is caused to hold in the next state. In particular, in the premise, a combination of the form $\neg f, \bigcirc f$ (or $f, \bigcirc \neg f$) may be used to mean that fluent f *becomes* true (resp., false). The language also includes constraints of the form $\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ where the l_i's are simple or temporal fluent literals.

In this language, default negation in clause bodies allows for the specification of *nondeterministic action laws*, of the form $[a](l_0 \vee \ldots \vee l_k) \leftarrow l_{k+1}, \ldots, l_m$, $not\ l_{m+1}, \ldots, not\ l_n$, stating that the execution of action a in a state in which $l_{k+1}, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold, makes nondeterministically one of $l_0, \ldots, l_k$ true. In fact, $[a](l_0 \vee \ldots \vee l_k) \leftarrow Body$ can be seen as a shorthand for the rules $[a]l_i \leftarrow Body, not\ [a]l_1, \ldots not\ [a]l_{i-1}, not\ [a]l_{i+1}, \ldots not\ [a]l_k$ $(i = 1, \ldots, k)$.

The laws above can be used to define persistency laws to deal with frame fluents as well as to complete the initial state in all the possible ways compatible with the initial state specification. The semantics of a domain description, is defined by extending the notion of *answer set* [11] to *temporal answer sets*, so to capture the linear structure of temporal models. We refer to [16] for details.

3 Declarative specification of business processes: merging annotations with data

A declarative specification of a business process can be given by exploiting the action theory above to define the effects of atomic tasks as well as their executability preconditions. This approach has been followed in different contexts such as in the declarative specification of web services in [26, 5] and in the declarative specification of agent communication protocols in [35, 14]. We show that causal laws have a relevant role in the specification of background knowledge, which is common both to the business process and to the business rules, and that the proposed approach allows for an easy integration of the data perspective.

The declarative specification of business processes has been advocated by many authors [32, 30, 25], as opposed to the more rigid transition based approach. A declarative specification of a process is, generally, more concise than transition based specification as it abstracts away form rigid control-flow details and does not require the order among the actions in the process to be rigidly defined.

The Temporal ASP language in Section 2 is well suited for defining immediate and indirect effects of atomic tasks and their preconditions. Consider, for instance, the business process of an investment firm in [7], where the firm offers financial instruments to an investor. The atomic task *investor identification* has as effect that the investor has been identified, while investor *profiling* has the nondeterministic effect that the investor is recognized as being either *risk_averse* or *risk_seeking*. This can be modeled by the action laws:

$[investor\_ident(I)]investor\_identified(I)$
$[profiling(I)](risk\_averse(I) \vee risk\_seeking(I)) \leftarrow investor\_identified(I)$

The first action law has empty precondition. The fact that $profiling$ can be executed only when the atomic task *investor identification* has been executed, can be modeled by introducing the precondition law:

$$[profiling(I)]\bot \leftarrow not\ investor\_identified(I))$$

which, literally, states that executing action $profiling$ in a state in which the investor I has not been identified gives an inconsistency. Observe that, in this language, an action is executable unless there is a precondition law for it whose antecedent is not satisfied. Hence, once the investor has been identified, the action $profiling(I)$ becomes executable. However, to guarantee that it will be eventually executed, we can add in $\mathcal{C}$ the DLTL constraint

$$\Box[investor\_ident(I)]\Diamond\langle profiling(I)\rangle\top$$

To force the execution of $profiling$ immediately after *investor identification*, instead, we could add the constraint: $\Box[investor\_ident(I)]\ \langle profiling(I)\rangle\top$.

The presence of DLTL constraints in a domain specification allows for a simple way to constrain activities in a business process. Observe that, as DLTL is an extension of LTL, it is possible to provide an encoding of all ConDec [28] constraints into our action language. The additional expressivity which comes from the presence of program expressions in DLTL, allows for a very compact encoding of certain declarative properties of the domain dealing with finite iterations. For instance, the property "action b must be executed immediately after any even occurrence of action a in a run" can be expressed by the temporal constraint: $\Box[(a;\Sigma^*;a)^*]\langle b\rangle\top)$, where Σ^* represents any finite action sequence.

In [7] it has been shown that program expressions can be used to model the control flow of a business process in a rigid way. However, the solution in [7] does not deal with non-structured workflows.

As concerns the data perspective, an atomic task which acquires the value of a data variable (data object) x can be regarded as an action assigning nondeterministically to x one of the values in its domain. Consider, for instance, the atomic task $verify\_status$ which verifies the status of a customer. Assume it has the effect of assigning a value (*gold, silver* or *unknown*) to a variable *status*. The task $verify\_status$ can be regarded as a non deterministic action assigning one of the possible values to the variable status:

$$[verify\_status](\ status(gold) \vee status(silver))$$

In general, we model a data acquisition task as a nondeterministic action. As an example, let us consider an atomic task *get_order* which acquires an order of a product P and an atomic task $select\_shipper(P)$ which selects a shipper among the available shippers, which are compatible with the choice of the product P. Let us introduce the notation $1\{[a]R(X) \mid P(X)\}1$ (similar to the notations used in Clingo and in S-models) as a shorthand for the two laws:

$$[a]R(X) \leftarrow\ not\ [a]\neg R(X) \wedge P(X)$$
$$[a]\neg R(X) \leftarrow\ [a]R(Y) \wedge P(X) \wedge P(Y) \wedge X \neq Y$$

meaning that after the execution of action a, $R(X)$ holds for a unique value of X among those values satisfying $P(X)$. Let $available\_product(P)$ and $available\_shipper(S)$ be the predicates defining the available products and shippers, and $compatible(P, S)$ be a predicate saying that product P and shipper S are compatible. We can represent the effect of action get_order by the law:

$1\{[get\_order]product(P) \mid available\_product(P)\}1$

and the effect of action $select\_shipper(P)$ as

$1\{[select\_shipper(P)]shipper(S) \mid available\_shipper(S)\}1$.

The requirement that P and S must be compatible can be enforced introducing the constraint:

$\bot \leftarrow [select\_shipper(P)]shipper(S) \wedge not\ compatible(P, S)$

meaning that it is not the case that the selected shipper S and the product P to be shipped are not compatible.

The above specification of the effects of the task $select\_shipper(P)$ has strong similarities with the specification of a post-condition for a service in [9]. Indeed, in [9], a post-condition of the form $R(\overline{x}) := \psi(\overline{x})$, associated with a service σ, requires that after the execution of σ the argument $\overline{x}$ of R is instantiated with a (unique) tuple $\overline{u}$ such that $\psi(\overline{u})$ holds in the previous state (artifact instance). As a difference with [9], where $\psi(\overline{x})$ is a first-order temporal formula, our temporal language does not allow for explicit quantification: all variables occurring in action and causal laws are intended to be universally quantified in front of the laws. Furthermore, in our approach we cannot deal with infinite domains. As usual in ASP, a finite groundization the set of laws in the domain specification is required. Abstraction techniques as those in [24] can be adopted to abstract infinite or large domains to a finite, small set of abstract values.

4 Specification of business rules: causality and commitments

The use of directional implications for modeling business rules as well as for modeling the conditional structure of norms is widely recognized in the literature [18]. In this section we claim that static and dynamic causal laws, proposed in the AI literature about reasoning about actions and change, are also appropriate for modeling business processes.

Consider the domain in examples 2 and 3 in [33], with the rule stating that if an insurance claim is accepted by reviewer A and reviewer B, then it is accepted. Suppose this is represented as the material implication

$$claimAccRevA \wedge claimAccRevB \supset claimAccepted$$

i.e., the clause $\neg claimAccRevA \vee \neg claimAccRevB \vee claimAccepted$. Suppose further, as in [33], that as a result of an action with direct effects, we accept models where such effects hold, that satisfy a background theory including the implication above, and, according to the Possible Models Approach [34], differ minimally from the previous state. Consider a state where $claimAccRevA$ already holds, and an action of acceptance for reviewer B occurs, with direct effect $claimAccRevB$. In order to satisfy the material implication, $claimAccepted$

should become true, or *claimAccRevA* should become false, or both; minimal difference with the previous state only excludes this third alternative, while providing equal status to the first two. If the redundancy in the process means that the assessment of a reviewer has no influence on the other's, then only the first result, where *claimAccepted* becomes true, is intended. The (static) causal rule

$$claimAccepted \leftarrow claimAccRevA, claimAccRevB$$

allows to obtain the first solution, given that its semantics imposes that in all states, if $claimAccRevA \wedge claimAccRevB$ is true (and, in particular, it just became true), then *claimAccepted* holds (and it becomes true as a side effect if the premise just became true).

However, the above implication might not actually be intended, as in case later steps in the process could make the claim not accepted. For example, the process model might specify that if the amount claimed is greater than a threshold, it should go through further approval by a supervisor (with possible effect $\neg claimAccepted$). Unlike [33], we consider the case where this does not mean that $claimAccRevA \wedge claimAccRevB$ should become false, i.e., at least one conjunct (or exactly one, for a minimal change) should become false. Rather, we suggest that here, after reviewers acceptance, *claimAccepted* actually stands for "accepted unless decision is overridden" Dynamic causal laws are suitable to represent this; the side effect of acceptance by the single reviewers becomes:

$\bigcirc claimAccepted \leftarrow \bigcirc claimAccRevA, \neg claimAccRevB, \bigcirc claimAccRevB$

$\bigcirc claimAccepted \leftarrow \neg claimAccRevA, \bigcirc claimAccRevA, \bigcirc claimAccRevB$

where syntactic sugar can be introduced, as in [8], to succinctly state that the conjunction $claimAccRevA \wedge claimAccRevB$ is *initiated* i.e., it becomes true.

Such rules correctly make *claimAccepted* true after reviewer acceptance, but, if a further step has the effect $\neg claimAccepted$, they do not "fire" because $claimAccRevA \wedge claimAccRevB$ is true, but it is not *becoming* true. Note the difference with the static causal rule which would fire (because $claimAccRevA \wedge claimAccRevB$ is true) and then contradict $\neg claimAccepted$.

A particularly significant case of the pattern above, where a fluent becomes true as an indirect effect of some activity, but may be canceled by further activities, is the one of **obligations**, which arise naturally in compliance rules: several such rules are variants of "if B happens, then A shall happen", or, "if B is (or becomes) true, then A shall become true". Compliance verification for such rules could be performed by verifying a straightforward representation of the rule as a temporal logic formula, e.g., in LTL, the formula $\Box(B \supset \Diamond A)$.

This, however, does not admit the possibility that a later activity cancels the obligation: e.g., if an order for goods is confirmed by the seller, goods have to be shipped; but if the customer cancels the order, the obligation to ship goods is canceled. An explicit representation of obligations is useful to this purpose. In this paper we limit our attention to one type of obligations in the classification in [19]: the case where a given condition should become true at least once, after they have been triggered; i.e., we consider achievement obligations in [19], and we only consider the case where the obligation should be fulfilled after it is triggered.

We then identify obligations with the notion of **commitment** from the social approach to agent communication [30, 20, 10]. A *(base) commitment* $C(i, j, A)$, means that agent i is committed to agent j to bring about A, while *conditional commitments* of the form $CC(i, j, B, A)$, mean that agent i is committed to agent j to bring about A, if condition B is brought about [35, 14]. In this paper we do not consider agents explicitly, and we concentrate our attention to base commitments $C(A)$ where A is a fluent; $C(A)$ is also a fluent, which can be made true, due to an action law or a dynamic causal law, as a direct or indirect effect of an activity in the process (order confirmation, in the example). The commitment (to ship goods, in the example) can be made false by an action with effect $\neg C(A)$ (the customer cancelling the order). Fulfilling the commitment (shipping goods) also makes the commitment false. Compliance verification, as we shall see in Section 6, amounts then to verifying that commitments, if introduced, are discharged, i.e., they are either fulfilled or explicitly canceled.

We refer to [7] for the treatment of defeasible business rules by means of default negation in ASP.

5 Translating business process workflows in ASP

The temporal action language introduced above provides a flexible and declarative specification language for business processes, and in [16] we have provided its translation to standard ASP.

There are, however, cases where the business process is naturally modeled (or it has already been modeled) in a workflow language such as YAWL [31]. In principle, such process models could be translated automatically to the temporal action language, but we have provided a direct translation to ASP for a subset of YAWL including AND- and XOR- splits and joins. The translation is based on an enabling semantics of arcs and tasks: an atomic task can be executed (i.e., the action can occur) when it is enabled. It is enabled when its only incoming arc is enabled, or it is an AND-join and all incoming arcs are enabled, or it is a XOR-join an one incoming arc is enabled. The execution of a task enables the outgoing arcs, and, in case it is a XOR-split, the execution of a subsequent activity based on the enabling of one such arc disables the other arcs.

6 Business process verification by bounded model checking

In [16] we have developed Bounded Model Checking techniques for the verification of DLTL constraints. In particular, the approach extends the one developed in [21] for bounded LTL model checking with Stable Models. The approach can be used for checking satisfiability of temporal formulas. To prove the validity of a formula, its negation is checked for satisfiability. In case the formula is not valid, a counterexample is provided.

Several verification tasks can be addressed within the proposed approach. Compliance verification (described in some detail in [7]) amounts to check that all

the business rules are satisfied in all the execution of the process. We distinguish among business rules which can be encoded as a temporal formula and business rules whose modeling involves commitments.

As an example of rule which can be encoded as a temporal formula to be verified, consider, in the order-production-delivery process in [24], the rule "Premium customer status shall only be offered after a prior solvency check": it can be verified by checking the validity of the temporal formula

$$\Box(solvency\_check\_done \vee \neg\langle offer\_premium\_status\rangle\top)$$

i.e., by verifying that in all executions of the business process if the action $offer\_premium\_status$ is executable, the fluent $solvency\_check\_done$ holds. As an example of rule modeled through causal laws whose effect is adding a commitment, consider the rule "if the investor signs an order, the firm is obliged to provide him a copy of the contract". It can be encoded by the causal law:

$$C(sent\_contract) \leftarrow order\_signed$$

We require that all the commitments generated are eventually fulfilled, unless they are explicitly cancelled (e.g., in the example, cancelling the order also cancels the obligation to send the contract). Observe that canceling a commitment would not be possible if the commitment to α corresponded directly to the temporal formula $\Diamond\alpha$. A commitment is also discharged when it is fulfilled, i.e., the following causal rule is added for all possible commitments:

$$\bigcirc\neg C(\alpha) \leftarrow C(\alpha) \wedge \bigcirc\alpha$$

Then the verification of rules involving commitments amounts to verifying the validity, for all possible commitments $\mathrm{C}(\alpha)$, of the formula:

$$\Box(\mathrm{C}(\alpha) \rightarrow \Diamond(\neg\mathrm{C}(\alpha)))$$

A verification task considered in [9] is that of verifying properties of a business process, under the assumption that the process satisfies some given business rules. This verification task can also be addressed in our approach: the specification of the business rules is given by adding temporal constraints (and, possibly, causal laws) to the domain specification. The executions of the resulting domain specification are then verified against other temporal properties.

Satisfiability and validity of a DLTL formula over the business process executions are decidable problems. However, given that BMC is not complete in general, an alternative approach to BMC in ASP is proposed in [15] to address the problem of completeness, by exploiting the Büchi automaton construction while searching for a counterexample.

7 Conclusions and related work

The paper presents an approach to the verification of the compliance of business processes with norms. The approach is based on a temporal extension of ASP.

The business process, its semantic annotation and the norms are encoded using temporal ASP rules as well as temporal constraints. Causal laws are used for modeling norms, and commitments are introduced for representing obligations. Compliance verification can be performed using the BMC technique developed in [16] for DLTL bounded model checking in ASP, which extends the approach for bounded LTL model checking with Stable Models in [21].

This paper enhances the approach to business processes compliance verification in [7] by taking into consideration the data perspective and by providing a declarative specification of the business process, while in [7] the control flow of a structured business process is modeled in a rigid way by means of a program expression. Also, we have shown that a direct encoding of the process workflow in ASP can be given and exploited for process verification.

Several proposals in the literature introduce annotations on business processes for dealing with compliance verification [12, 18, 33]. In particular, [18] proposes a logical approach to business process compliance based on the idea of annotating the business process. Annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL), which combines defeasible logic [3] and deontic logic of violations [17]. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by task execution. Ad hoc algorithms for propagating obligations through the process graph are defined.

The idea of describing the effects of atomic tasks on data through preconditions and effects is already present in [23], where effects and preconditions are sets of atomic formulas, and the background knowledge consists of a theory in clausal form; I-Propagation [33] is exploited for computing annotations. In our approach the domain theory contains directional causal rules rather than general clauses (which allow unintended conclusions to be avoided when reasoning about side effects), and domain annotations are combined with data properties in a uniform approach. In the related paper [33] several verification tasks are defined to verify that the business process control flow interacts correctly with the behaviour of the individual activities.

In [9] a service over an artifact schema is defined as a triple: a precondition, a post-condition and a set of static rules, which define changes on state relations, and are formulas in a first-order temporal logic. State update rules $S(\overline{x}) \leftarrow \phi^+(\overline{x})$ and $\neg S(\overline{x}) \leftarrow \phi^-(\overline{x})$ are essentially specific kind of causal laws whose antecedents ϕ^+ and ϕ^+ are evaluated in the artifact instance in which the service is executed and whose consequents are added to the resulting artifact instance. [9] identifies a class of *guarded* artifacts for which verification of properties in a (guarded) first-order extension of LTL is decidable. While our action language does not allow for explicit quantification, it allows for a flexible formulation of action effects and causal laws, which permits (as shown in Section 3) an encoding of post-conditions as in [9].

In [4] compliance checking for BPMN process models is based on the BPMN-Q visual language. Rules are given a declarative representation as BPMN-Q queries, which are translated into temporal formulas for verification.

In [25] the Abductive Logic Programming framework SHIFF [2] is exploited in the declarative specification of business processes as well as in the verification of their properties. In [1] expectations are used for modelling obligations and prohibitions and norms are formalized by abductive integrity constraints.

In [29] Concurrent Transaction Logic (CTR) is used to model and reason about general service choreographies. Service choreographies and contract requirements are represented in CTR. The paper addresses the problem of deciding if there is an execution of the service choreography that complies both with the service policies and the client contract requirements.

Temporal rule patterns for regulatory policies are introduced in [13], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The approach is used essentially for event monitoring.

References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, and G. Sartor. Mapping of Deontic Operators to Abductive Expectations. *NORMAS*, pages 126–136, 2005.
2. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Trans. Comput. Log.*, 9(4), 2008.
3. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2:255–287, 2001.
4. Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic, LNCS 5240. In *BPM*, pages 326–341. Springer, 2008.
5. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, pages 288–304, 2007.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
7. D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. Theseider Dupré. Verifying business process compliance by reasoning about actions. In *CLIMA XI*, pages 99–116, 2010.
8. M. Denecker, D. Theseider Dupré, and K. Van Belleghem. An inductive definitions approach to ramifications. *Electronic Transactions on Artificial Intelligence*, 2:25–97, 1998.
9. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.
10. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *AAMAS03*, pages 520–527.
11. M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.
12. A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.
13. C. Giblin, S. Müller, and B. Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. *IBM Reasearch Report*, 2007.
14. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5:214–234, 2007.

15. L. Giordano, A. Martelli, and D. Theseider Dupré. Achieving completeness in bounded model checking of action theories in ASP. In *Proc. KR 2012*.
16. L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 2012.
17. G. Governatori and A. Rotolo. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. *Australasian Journal of Logic*, 4:193–215, 2006.
18. G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.
19. Guido Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law*. IEEE, 2010.
20. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems, Springer LNAI 2650*, 2003.
21. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
22. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
23. J. Hoffmann, I. Weber, and G. Governatori. On compliance checking for clausal constraints in annotated process models. *Information Systems Frontieres*, 2009.
24. D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam. On enabling data-aware compliance checking of business process models. In *Proc. ER 2010, 29th International Conference on Conceptual Modeling*, pages 332–346, 2010.
25. M. Montali, P. Torroni, F. Chesani, P. Mello, M. Alberti, and E. Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundam. Inform.*, 102(3-4):325–361, 2010.
26. S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. 11th Int. World Wide Web Conference, WWW2002*, pages 77–88, 2002.
27. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428445, 2003.
28. Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops, LNCS 4103*, pages 169–180. Springer, 2006.
29. D. Roman and M. Kifer. Semantic web service choreography: Contracting and enactment. In *International Semantic Web Conference, LNCS 5318*, pages 550–566, 2008.
30. M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.
31. A. M. ter Hofstede, W. M. P. van der Aalst, M. Adamns, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. 2010.
32. Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*, 2006.
33. I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases (DAPD)*, 2010.
34. M. Winslett. Reasoning about action using a possible models approach. In *Proc. AAAI 88, 7th National Conference on Artificial Intelligence*, pages 89–93, 1988.
35. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. *AAMAS'02*, pages 527–534, 2002.