# Detecting Compliance with Business Rules in Ontology-Based Process Modeling

Carl Corea[1], Patrick Delfmann[1]

[1] University of Koblenz-Landau, Institute for IS Research, Koblenz, Germany
{ccorea,delfmann}@uni-koblenz.de

**Abstract**. Extending business processes with semantic annotations has gained recent attention. This comprises relating process elements to ontology elements in order to create a shared conceptual and terminological understanding. In business process modeling, processes may have to adhere to a multitude of rules. A common way to detect compliance automatedly is studying the artifact of the process model itself. However, if an ontology exists as an additional artifact, it may prove beneficial to exploit this structure for compliance detection, as it provides a rich specification of the business process. We therefore propose an approach that models a rules-layer ontop of an ontology. Said rules-layer is implemented by a *logic program* and can be used to *reason* about the compliance of an underlying *ontology*. Our approach allows ad-hoc access to external ontologies, other than similar approaches that are reliant on a redundant logical representation of process model elements.

**Keywords**: Compliance Management, Business Rules, Business Process Models, Business Ontologies

## 1. Introduction

*Compliance management* is an important part of business process modeling (BPM), aimed to ensure that the company practices which can be entailed from the respective process models are compliant to regulations and business rules [13]. This especially holds for sectors subject to a high degree of regulatory control, such as the financial industry or healthcare [2], [13]. As an example, vendors of financial services may want to warrant that their process for granting loans does not violate any laws or obligations. Compliance management therefore supports *improving* business processes, as potential violations can be eliminated after they have been found, e.g. through re-modeling the business process [3].

The necessity for compliance management has yielded the rise of *automated* approaches, as trying to investigate large company processes for compliance violations *manually* can be seen as an unfeasible task for humans [13]. Following [10], a core notion of such automated approaches is the study of the business process model itself. Such business process models are typically represented through graphical modeling languages such as Event-driven Process Chains (EPC) [26], in order to provide a suitable balance between specification and readability. However, using such languages

13th International Conference on Wirtschaftsinformatik,
February 12-15, 2017, St. Gallen, Switzerland

Corea, C.; Delfmann, P. (2017): Detecting Compliance with Business Rules in Ontology-Based Process Modeling, in Leimeister, J.M.; Brenner, W. (Hrsg.): Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI 2017), St. Gallen, S. 226-240

to model business processes, as well as limiting compliance management to this artifact, can bare restrictions in regard to the semantic interpretation of the process model [25]. Although languages such as EPC offer some guidelines towards how to encode process syntax and semantics, the content of the models ultimately lies in the responsebility of the process modeler [21]. Especially when collaboratively creating process models, this can result in different interpretations of the process semantics due to problems such as ambiguity in human language [9], [25]. These different interpretations of the process model can pose potential problems, if they are meant to be analyzed as a central corpus in the scope of compliance management [7], [25].

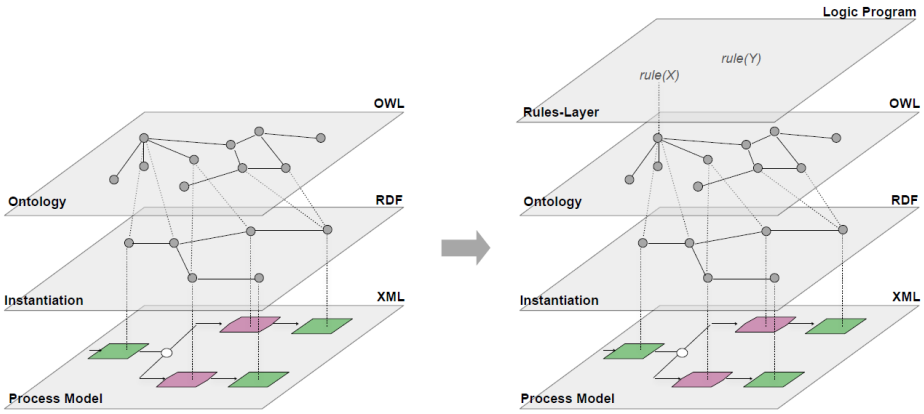


**Figure 1.** 4-layered framework adapted from [25]

To conquer the problem of different interpretations of business processes, works such as [7-8], [20], [24-25] have proposed to use ontologies to create a shared terminological and conceptual understanding of process models. Ontologies, which Gruber [16] defines as a formal and shared specification of a domain of interest, are a central object of interest in scientific fields such as the Semantic Web [18], which is why works such as [7-8], [20], [24-25] have proposed it may prove as beneficial to investigate applying this object for business process modeling. A main advantage of using ontologies in BPM is that a process model is extended such that machines can access it in a way useful for humans [18]. To this aim, elements of the process model can be annotated to ontology concepts, which is also referred to as *semantic annotations*. This promotes the understanding of the overall business process, as the process model is linked to a conceptual and terminological understanding shared by the modelers. Following [25], ontologies can be furthermore extended by the modelers. For instance, relevant policies or business rules can directly be included in such an ontology, explicitly *specifying the relations between the business process and such regulations*. Concluding, this report investigates utilizing such an ontology as an artifact for compliance management, as an ontology can be considered as an advanced basis for this form of process-oriented compliance management.

This is clarified in figure 1. On the left, which shows the three-layered model for a semantic annotation by [25], we see a business process which has been related to a

business ontology. The lower layer represents the classical business process - in this case an EPC diagram. The upper ontology comprises terminological concepts relevant for the respective business as well as their relations. The intermediate instantiation is used to assign elements of the business process to ontology concepts. I.e. instances of the business ontology are used to define the semantics of the EPC model elements. In this paper, we aim at providing an approach which layers a specification of business rules ontop of a business ontology. We refer to our approach as a 4-layered framework, meaning that our contribution extends the existing framework by [25]. This can be seen on the right of figure 1, which shows a rules-layer that can access the underlying ontology. There has been recent attention on combining rules and ontologies in the field of the Semantic Web, due to the complimentary characters of these components [23]. While there is no clear standard yet on how to combine these components, there is a consensus in literature that logic programs can be used to express rules over underlying ontologies [6], [15].

As a result, logic programs can be applied to formalize business rules which can then directly access the vocabulary used in the underlying ontology. Due to the fact that the mentioned ontology is connected to the business process itself, the business rules expressed in the logic program can also access the actual business process itself. Thus, the approach proposed in this report allows to apply the amenities of logic program reasoning in the context of compliance management. By layering rules ontop of already existing business ontologies, detecting compliance is not limited to analyzing the artifact of the business process model itself, but rather the more sophisticated description present in the business ontology can be exploited for verifying compliance in business processes.

The contributions of this report can be summarized as follows. At first, we show that our framework is a novel approach on combining business rules and business ontologies by motivating the 4-layered framework in the context of related work in section 2. Furthermore, after providing a brief recap on how to create semantic annotations for business processes, we show how the business ontology can be integrated into a logic program in order to ensure that company processes adhere to business rules and regulations in section 3. We illustrate our approach and also provide a demonstration in section 4. Finally, our discussion is concluded in section 5.

## 2. Related Work and Motivation

There have been numerous proposals for automatedly detecting compliance of processes with regulations or business rules [10]. One major school of thought are graph-based approaches [7], [10]. Here, graph-patterns which represent business rule violations are defined o graphically modelled, e.g in approaches like BPMN-Q, eCRG or DMQL[7]. Consequently, a pattern search can be applied to the graph-structure representing the business process in order to find respective violations. It is important to realize that mentioned graph-based approaches focus on analyzing the artifact of the business process model itself.

As mentioned, process models can be linked to business ontologies. In result, next to the artifact of the business process itself, the company may have a second artifact of a business ontology, which can be used as a basis for compliance management. This has been proposed by works such as [7-8][20][24-25], due to the sophisticated semantic structure offered by ontologies. Our approach is therefore an extension to works such as [25], which tries to capture the possibilities that are potentially present.

A main concept of our approach relates to defining business rules as logic expressions. Many others have already proposed using logic expressions instead of graph-based approaches. For example, there is a broad consensus in academia, that temporal logic is suitable to check process models against business rules [10]. While we do not disagree with this claim, we would like to point out that there are limitations of using temporal logic for this aim that have been identified by works such as [10]. For example, investigating process elements with complex annotations or dependencies can be seen as a very complex task [7], [10]. We therefore argue, that using temporal logic for compliance checking should not be taken as self-evident. Authors like Gruhn and Governatori also agree that such formalisms rooted in temporal logic may suffer from some limitations in compliance checking and have therefore proposed other families of logic for this use-case [13], [17].

Said authors have shown that logic programs can be used to validate syntax or compliance in business processes. Following [22], using logic programs to verify business rules is applied as follows: At first, a *new* logic program is derived from a process model. To clarify, the logic program is independent of the original process model. All model elements and their relations are *redundantly* translated into a logic representation. Only then can the logic program reason about compliance. In our approach, we propose to layer a logic program ontop of the already existing process ontology. In this way, process elements do not have to be redundantly translated into a logic representation of an independent logic program, but rather business rules in the form of logic expressions can directly access the underlying ontology and reason about the compliance of the ontology, respectively the business process itself. In case of changes to the business process or the business ontology, our approach is therefore still able to verify compliance without the effort of having to repeat a redundant translation of changed elements.

Works such as [6], [15] have investigated integrating logic programs and ontologies. Said works are however for the Semantic Web and are not specifically aimed at business process management. [8] have also proposed an ontology-driven approach to detect compliance with rules. Here however, they do not investigate using logic programs in order to express regulations and rules.

To the best of our knowledge, our approach is the first to study the intersection between *(a) using rules to detect compliance based on the artifacts of both a process model and a business ontology,* and *(b) using logic programs to implement these rules.* The framework therefore allows to exploit logic program reasoning relative to a business ontology, without having to translate elements of the business ontology into a redundant logical representation in a respective logic program. Table 1 positions our approach in the above mentioned intersection.

**Table 1.** Approach research gap

| *Literature* | *Uses logic programs to detect compliance* | *Logic programs can access underlying artifacts in an ad-hoc manner* |
|---|---|---|
| [1][13-14] [17][22] | x | |
| [6][12][15] [19][23] | | x |
| Proposed Approach | x | x |

## 3. Layering Rules Ontop of Business Process Ontologies

This section introduces modeling a rules-layer ontop of a business ontology.

### 3.1. Ontology-Based Process Modeling

Following [25], the scientific results in regard to extending information with semantic annotations can successfully be applied to BPM. Said authors employ a three-layered approach, combining the actual *business process model* with a *business ontology* through an intermediate *instantiation*, as can be seen in figure 1.

The ontology classes define terminological knowledge relevant to the company. For example, *entities* such as organizational units, tasks, events, services or rules and their individual *relations* can be modeled. It is important to realize, that companies do not necessarily have to model such an ontology themselves. Works such as [20] have already proposed reference ontologies that can be re-used and adapted to individual company requirements. In this work, we assume that the ontology is stored in the web-ontology language format (OWL)[1], which is the W3C standard for knowledge representations. Hence, next to the already introduced *terminological* knowledge, *axiomatical* instances of ontology concepts can be created. This is shown in figure 2, which provides an exemplary business ontology. One can observe that this ontology is subdivided into classes (e.g. Unit) and instances (e.g. Production and Sales), whose relationships are defined by this OWL graph-structure.

As a next step, instances of the business ontology are used to define the semantics of process model elements. We want to emphasize that this approach can be applied to arbitrary process models [25].

A process model is defined by element types and specific elements of such types [4]. The ontology can therefore be used to define the semantics of element types, also referred to as *language constructs*, and elements, also referred to as *model elements*. Language constructs, such as events, functions or connectors, should be modeled in the

[1] https://www.w3.org/TR/owl2-syntax/

reference ontology as can be seen in figure 2. As mentioned, the language constructs represented in OWL can be furthermore extended to fit individual company needs. For instance, EPC events could be specialized to create unique and distinguishable event types relevant to the company. Next, model elements can be represented in the ontology through an instantiation of the previously defined language construct classes. Figure 3 illustrates mapping an EPC diagram to a business ontology excerpt. As can be seen, the ontology comprises all relevant EPC language constructs, represented through the respective ontology classes. Every model element resided in the business process model is assigned to an ontology instance. In this way, ontology instances represent the individual model elements from the viewpoint of the business ontology [25]. As a main result, this approach has generated a conceptual viewpoint for the EPC diagram by making the process model accessible in the ontology. This viewpoint could already be exploited to pose conceptual queries regarding the business process. Assuming an OWL ontology, the W3C query language SPARQL[2] can be employed to answer such queries [18].

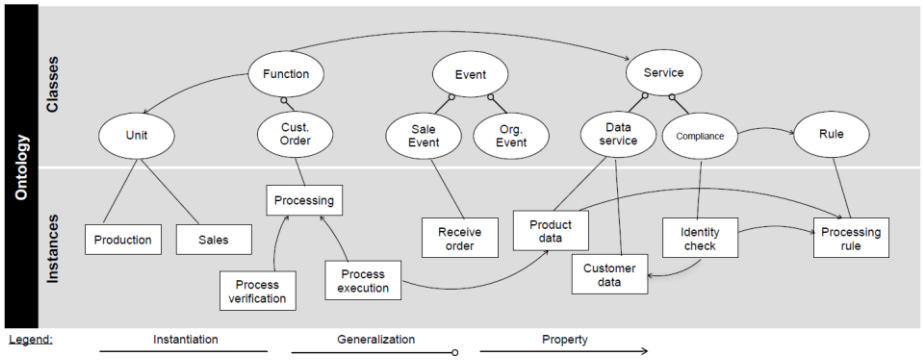


**Figure 2.** Exemplary business ontology

The model elements of the EPC diagram can now be assigned to further ontology concepts in order to create richer semantic annotations and therefore foster the semantic understanding of the business process, as individual model element semantics and their relations can be explicitly defined. Following [25], a major advantage of this approach is that copious constructs offered by the OWL formalism can be exploited to define formal semantics of great granularity. For example, the possibility to model generalizations or properties such as transitivity, symmetry or inversion between OWL instances allows to use sophisticated reasoning capabilities to analyze process models.

Business ontologies can be modeled or extended according to company needs. Consequently, business rules or regulations and their relation to the business process can be incorporated in the ontology. In result, the business ontology can be seen as an advanced artifact to use in the scope of compliance management [7], [25]. This leads us to our proposal, namely to extend the framework by [25] by modeling a rules-layer ontop of the ontology-layer. This rules-layer should offer the possibility to express

[2] https://www.w3.org/TR/rdf-sparql-query/

business rules and regulations relative to the business ontology and verify their compliance accordingly. To this aim, we propose to utilize a logic program formalism in order to implement said rules-layer, which we introduce subsequently.

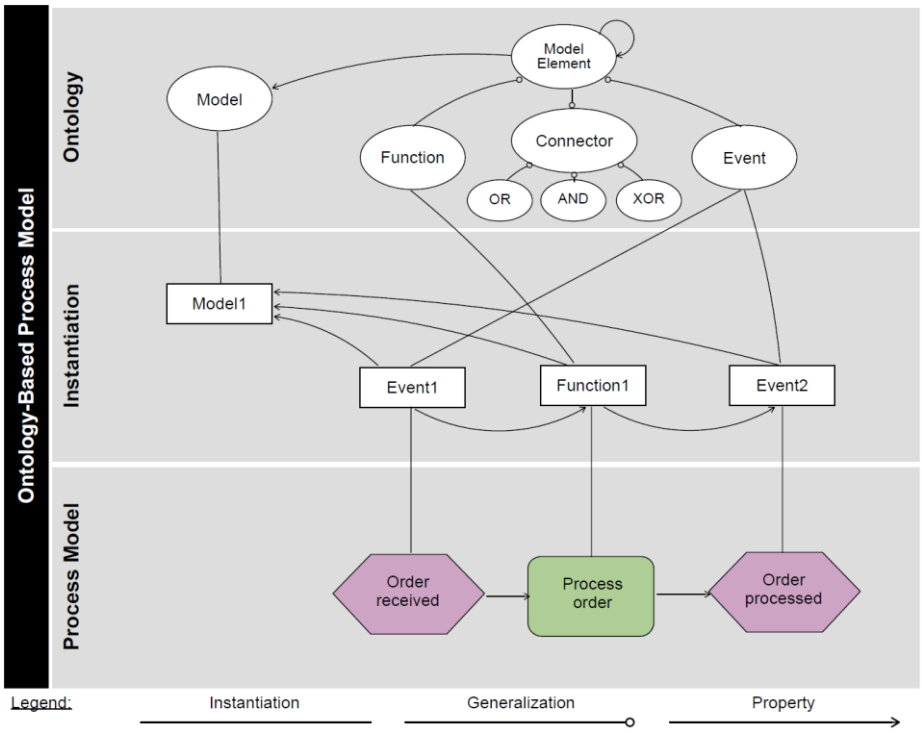


**Figure 3.** Mapping an EPC diagram to an ontology

### 3.2. DL-Programs

The Semantic Web architecture allows to model rules ontop of knowledge representations, i.e. ontologies [18]. In this way, the complimentary characters of these two layers can be used to promote automated processing mechanisms. As an important design choice, the rules-layer and the ontology-layer should be interoperable but abstracted from each other, as dictated by the Semantic Web architecture [18]. While there is no clear standard yet on which technologies to use for combining rules and ontologies, there have been several proposals [6].

What is important in the context of this work, is that research suggests that the mentioned rules-layer can be implemented by logic programs [15]. To clarify, logic programs can be used to express rules that mention the vocabulary of an external ontology, i.e. a description logic knowledge base. Thus, the sophisticated reasoning possibilities offered by logic programming can be used to infer information regarding ontologies - more specifically business ontologies. Again, as there are many families of logic, many proposals have been made as to which form of logic programs to use in

order to express rules for underlying ontologies. For a detailed survey, please see [15]. As a design choice, we have chosen an approach by [6] entitled *DL-programs*. DL-programs are a formalism able to extend logic programs with description logic expressions. More specifically, they allow to combine normal programs and description logic ontologies, respectively answer-set semantics with first-order semantics. Therefore, they meet our requirement of being able to express business rules and regulations relative to a business ontology.

**Recap of Logic Programming.** To recall logic programs in general, a logic program is defined as a tuple $\boldsymbol{t}$ = **(P,C)**, where **P** is a set of predicate symbols and **C** is a set of constants [23]. The rudiments of this signature can be used to express *rules*. Such a rule *r* consists of a *premise* and a *conclusion* of the general form

$$head \leftarrow rule \tag{1}$$

meaning that the head, or conclusion, of *r* is true, if the body of the rule is satisfied [23]. If the body of *r* is empty, *r* is referred to as a simple fact. The head and body of the rule can be used to compose formulas of the form

$$h \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, b_m \tag{2}$$

where each *h*, $a_i$ and $b_i$ are so-called *atoms*. Such an atom is defined as $p(t_1, \ldots, t_m)$, where *p* is a predicate symbol of **P** and every $t_i$ is either a constant from **C** or a simple variable, the latter denoted by a capitalized character. Note that every $a_i$ is a so-called positive atom, and every $b_i$ is a so-called negative atom, indicated by the *not*.

As an example, the following logic program in figure 4 could be used by a financial service to define rules regarding account values.

```
(i)    account(a, 100)
(ii)   account(b, -100)
(iii)  positveBalance(A) ←  account(A,B), B > 0
(iv)   error(A) ←  account(A), not positiveBalance(A)
```

**Figure 4.** Exemplary logic program

The first two lines of this logic program are simple facts, stating two accounts entitled *a* and *b* and their respective account balance. The rule in (iii) is used to verify if an account has a positive balance. Here, the capital *A* and *B* represent variables, respectively the account name and account balance. The body of this rule, i.e. the premise, is satisfied if *B* is positive, meaning that only in this case *positiveBalance(A)* could be concluded. A customer account may be required to be of positive balance due to business requirements. Hence, the rule in (iv) models a violation of this requirement. The head of this rule - *error* - is true as soon as there exists and account which is not of *positiveBalance*. As the account named b represents such a case, the depicted logic program can be used to entail that an error is present.

**Answer-Set Semantics.** The use of variables allows to entail that an error is present for account *b*. This is an example of so-called *answer-set* semantics of logic programs [23]. In early research on logic programs, variables were not included, meaning that rules consisted of simple forms similar to a ←b. Such logic programs can be used to

entail so-called *well-founded* semantics [11], which can be seen as simple *proofs*. Other than such simple proofs, a paradigm shift to answer-set semantics, which can be traced back to the works of Gelfond and Lifschitz [11], has allowed to define *answer-sets*. These answer-sets can be understood as a model satisfying a logic program. Consequently, an answer-set *M = { account(b,-100) }* can be derived from the exemplary logic program. As a result, the financial service could benefit from such an answer-set to identify specific accounts violating the business rule regarding a positive account balance [11].

**Introduction to DL-Programs.** Continuing our exemplary logic program, computing an answer-set was limited to facts and rules *contained* in this logic program. In order to allow the logic program, i.e. the rules, to reason about information in external ontologies, they have to be extended in such a way that they can *access* said external knowledge bases.

DL-programs [6] represent such an extension of logic programs that allow to access vocabulary of underlying ontologies. A DL-program consists of a normal program *P* and an external knowledge base, i.e. ontology, *L* [6]. While *P* is a finite set of rules based on predicates and constants as introduced, the ontology comprises concepts, roles and individuals. In result, such a DL-program can use a business ontology as a knowledge base *L* and layer a logic program *P* ontop of it.

A question that may arise is how exactly *P* can be extended to access concepts, roles and individuals in *L*. To this aim, [6] have proposed to extend the rules of *P* with so-called *DL-atoms*. While it is not our intent to elaborate on the syntax of such DL-atoms in great detail, it is sufficient to realize that DL-atoms are of the general form:

$$DL[Q](t) \tag{3}$$

Inspecting this complex more closely, the sequence *DL* signalizes the beginning of a DL-query. One can observe that such a DL-query consists of *Q*, which may refer to a *concept* or *role* of the knowledge base *L*. The *(t)* is a simple logic program *term* as introduced earlier, i.e. a constant or a variable. By extending the logic program *P* with the DL-query *DL[Q]*, it can refer to a vocabulary *Q* of an external ontology. As an example, figure 5 shows a DL-program based on the following logic program *P* and knowledge base *L*:

```
L:
(i)       event ⊑ modelElement
(ii)      event(e1)
(iii)     event(e2)

P:
(iv)      evt(X) ⟵ DL[event](X)
```

**Figure 5.** Exemplary DL-program

In this example, a business ontology *L* contains terminological knowledge about the language construct *event*, as well as axiomatical *event* instances. The logic program rule in (iv) shows how a DL-atom is used to extend a logic program rule. It is important

to realize, that *event* in square brackets of the DL-atoms in (iv) refers to the *event* concept of the business ontology *L*. I.e. knowledge contained in the ontology *L* does not have to be *redundantly* expressed as a fact in *P*, but rather the DL-program allows a logic program *P* to access an ontology *L* in an *ad-hoc* manner. To conclude, regarding the logic program *P*, an answer-set *M = { evt(e1), evt(e2) }* could be directly entailed by the means of using (iv) to match the variable *X* in the head of the rule with the variable *X* in the DL-atom. This answer-set could then be processed in additional rules of *P*.

As can be seen in the example, DL-programs allow logic programs to access the information stored in an external business ontology. This facilitates powerful and expressive ways to process knowledge bases by the means of rule bases formalized through logic programs [6]. In our opinion, enabling a rules-layer to access a business ontology in an ad-hoc manner is a stronghold of our approach which we discuss consequently.

## 4. Compliance Checking Approach

This section demonstrates how DL-programs can be used to express business rules in order to detect compliance based on both the artifacts of the business process model as well as a business ontology.

### 4.1. Framework Architecture

**Creating an Ontology-Based Process Model.** Figure 6 provides an overview of the framework architecture. The proposed approach extends the three-layered model by [25]. On the left, the process model is connected to a business ontology by instantiation. We denote this as an *ontology-based process model*.

Next, the rules-layer consists of a logic program expressing business rules. The specific business rules may originate from business requirements or external regulatory policies. The rules-layer can directly access the ontology using the DL-program formalism. The rules of the logic program can therefore infer information about the entire ontology-based process model, as the ontology is connected to the process model. The proposed approach is therefore capable to detect whether a process model complies with business rules. This is depicted by the compliance detection component. Here, the DL-program is applied to infer answer-sets of compliance violations. To clarify, these answer-sets consist of *ontology-instances* violating rules of the logic program. These answer-sets therefore also reflect specific *process model elements* violating these logic program rules, respectively business rules. These sets of process model elements can consequently be browsed by the modeler and remodeled according to rules and regulations.

A barrier for the implementation of our approach is the necessity for the artifact of an ontology-based process model. Companies must annotate their processes to an ontology. While a company could perform this task manually, there are existing approaches showing that this task can be supported automatically [5]. In [5], the authors show that identifiers of processes and ontologies can be terminologically standardized and

thus matched accordingly. This lowers the effort that has to be invested by companies. Undoubtedly, the initial creation of an ontology-based process model has to be considered by companies. However, in our opinion, modeling company processes geared towards a business ontology helps to create a shared understanding across the entire organization. On the long-term, this can be seen as beneficial for the scalability and maintenance of business processes [8].

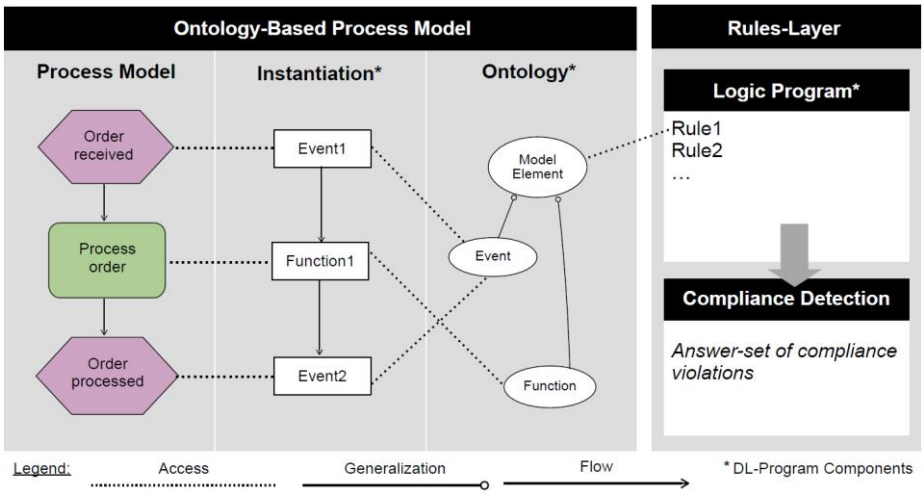


**Figure 6.** Framework architecture

**Expressivenes.** While temporal logic can be used to verify compliance, it still suffers from limitations [13], [17]. The expressiveness of our approach therefore aims to conquer some of these limitations, while not sacrificing any amenities. This is achieved through the instantiation. As shown in figure 6, every process model element is represented by an ontology instance. All flows between process model elements are also captured in the instantiation. This means, the execution semantics of the process is encoded in the ontology and can be processed accordingly. In result, sequences, loops or gateways can also be processed by our approach. Van der Aalst et al. [27] have categorized different types of compliance rules. It is beyond the scope of this report to discuss this categorization, but it is used here to specify the expressiveness of our approach in relation to said categorization. So far, we have successfully implemented rules of the categories *existence*, *precedence*, *chain precedence*, *response*, *between*, *exclusive*, *mutual exclusive*, *inclusive*, *prerequisite* and *corequisite*. A clear limitation are cardinality restraints or parallel processes. For further details on the DL-formalism, please see [15], as this paper introduces the syntax of DL-queries that are the foundation of compliance checking in our approach.

### 4.2. Demonstration

To demonstrate our approach, the following exemplary scenario was implemented. We envision a scenario where a company wishes to apply our approach to ensure a business

process complies to a business rule. For simplicity, rules and business process will be kept minimal. We assume that the company conducts the task of *paying a bill* within their process. It is furthermore assumed, that a corresponding business rule demands that during this process the bill is *checked* before it is paid. Assuming that the exemplary company aims to create ontology-based process models, figure 7 depicts artifacts which can be utilized in the scope of compliance management for our scenario. Given that a modeler has created the process model in figure 7 (i), our approach allows to detect whether this model complies with the mentioned business rule as follows.

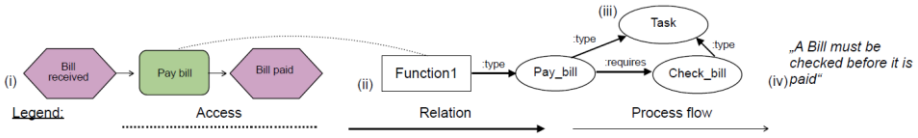


**Figure 7.** Artifacts available for the compliance management scenario

In (iii), the ontology is depicted as an OWL graph-structure. The company has modeled an ontology class *Task*, which is specialized into *Check_bill* and *Pay_bill*. It is important to realize that these are not instantiations, but simply a specification of the two concepts, which are *tasks*. An instantiation is shown in (ii). The ontology in (iii) was extended by a business rule, indicating that the concept of *Pay_bill* requires *Check_bill*. In this context, the edge labelled *requires* encodes the business rule in (iv). Intuitively, the process model in (i) does not comply to the company policy in (iv), as the bill is not checked before it is paid. The company can implement a DL-program based on a logic program encoding (iv) and the business ontology in (ii)-(iii). Figure 8 depicts this DL-program. For simplicity, namespaces for standard W3C vocabulary such as *OWL:class* are omitted. In the ontology, the class of a *Task* is defined. This is performed analogously for *Check_bill* and *Pay_bill*. Figure 8 also shows how a process model element can be serialized as XML exemplarily. In our scenario, as the modeler has only modeled the task *Pay_bill*, we may only create this single instantiation. The *required* relation, meaning that *Pay_bill* requires *Check_bill*, is also serialized. A logic program can then be layered ontop of this serialization, as shown in figure 8. In the first line of this logic program, an ad-hoc access to instances of the ontology is defined. The *task* in the head of the rule is a logic program predicate, whereas the *Task* in the body of the rule directly refers to the ontology concept by the means of the DL-atom. Line two of the logic program shows a specification of the aforementioned business rule. Here, we can *conclude* an error, if there exists an instance *X* requiring an instance *Y*, and there is no such instance *Y*. The DL-atom in the body of the rule also directly mentions the *required* vocabulary of the ontology. Thus, this relation already present in the ontology can be accessed directly as opposed to being redundantly represented in the logic program. In result, the logic program can correctly infer that an error is present, through the answer-set *M = { error("Pay_bill", "Check_bill") }*. Such inferences can not only be drawn theoretically, but also by many logic program reasoners such as RACER or Hermit [7]. [6] provide a web-interface[3] allowing to enter DL-programs. The serialization as shown in figure 8 can be entered in this web-interface in order to

[3] https://www.mat.unical.it/ianni/swlp/

conclude that mentioned error is present. In our opinion, this shows that our approach can successfully be applied to implement our scenario, namely to detect wether a process model complies to a business rule. Given the artifact of an ontology-based process model and a logic program encoding rules and regulations, the technology to conduct compliance management on the basis of our approach is ready to use.

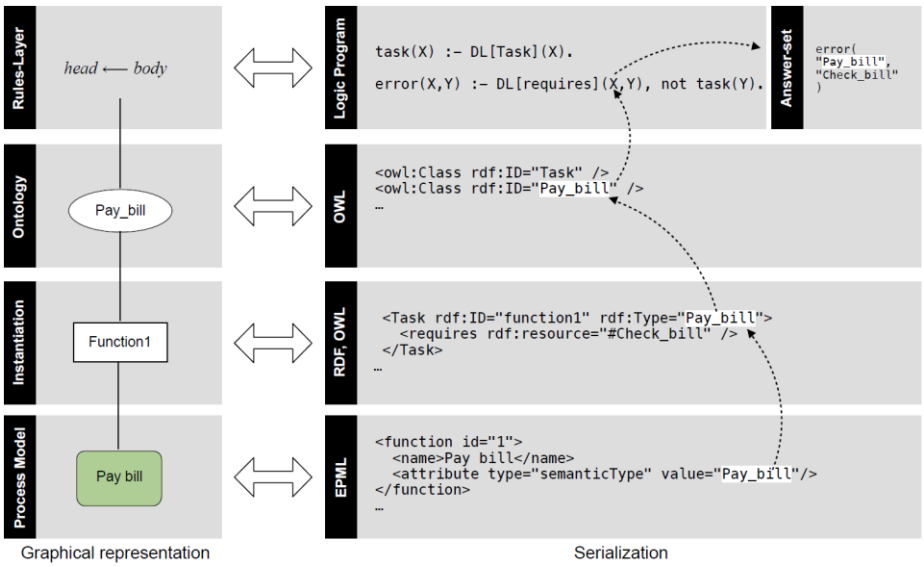


**Figure 8.** DL-program to entail process model elements that violate business rules

Our example covered a rule that checks existence. This can be extended to model different types of compliance rules, as mentioned in the subsection on expressiveness. For example, one could verify that the *Pay_bill* function is executed between two events, or that a function A is limited in precedence over a function B.

## 5. Conclusion and Outlook

The contribution of this paper is an approach capable of verifying if a business process complies with business rules, based on analyzing the artifact of a so-called ontology-based process model. The proposed approach allows to (a) specify business rules as logic program expressions relative to an external business ontology, (b) utilize logic program reasoning to find process model elements violating these rules and (c) access information stored in the business ontology directly, i.e. without a redundant transformation of ontology-instances into a logic program representation.

An exemplary implementation of our approach shows that our approach can be successfully applied to find sets of process model elements violating business rules. Future work is to be directed to apply our approach to large-scale process models and business rules. However, the success of using DL-programs for Semantic Web data-

sets [6] leads us in our belief, that applying our approach to business related data-sets should pose no significant computational problems per se.

A clear limitation of our approach is, that it is dependent of (1) a business ontology connected to a process model and (2) a logic program rule base. Following [7], implementing these artifacts is not yet significantly performed in practice. Literature however strongly suggests the potential of using these artifacts for process modeling and compliance management [7], [13], [20], [22], [24-25]. We therefore see great research potential in assisting companies to create and manage these artifacts. Works such as [5] show that these tasks can be supported automatedly.

As a conclusion, incorporating semantics in the scope of compliance management can contribute towards finding violations in business processes and therefore aid the improvement of company process. Here, using logic program techniques to *reason* about business *ontologies* assists the automated detection of compliance violations. Our approach, allowing an ad-hoc access for rules relative to a process model, lowers the effort that has to be directed towards this aim by companies.

## References


1. Antoniou, G., & Arief, M. (2001). Executable Declarative Business rules and their use in Electronic Commerce. Intelligent Systems in Accounting, Finance and Management, 10(4), 211-223.
2. Becker, J., Delfmann, P., Dietrich, H. A., Steinhorst, M., & Eggert, M. (2016). Business process compliance checking–applying and evaluating a generic pattern matching approach for conceptual models in the financial sector. Information Systems Frontiers, 18(2), 359-405.
3. Delfmann, P., & Höhenberger, S. (2015). Supporting Business Process Improvement through Business Process Weakness Pattern Collections. In 12th International Conference on Wirtschaftsinformatik (pp. 4-6).
4. Delfmann, P., Steinhorst, M., Dietrich, H. A., & Becker, J. (2015). The generic model query language GMQL-Conceptual specification, implementation, and runtime evaluation. Information Systems, 47, 129-177.
5. Delfmann, P., Herwig, S., & Lis, L. (2009). Unified enterprise knowledge representation with conceptual models-Capturing corporate language in naming conventions. ICIS 2009 Proceedings, 45.
6. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. Artificial Intelligence, 172(12), 1495-1539.
7. Fellmann, M., Delfmann, P., Koschmider, A., Laue, R., Leopold, H., & Schoknecht, A. (2015). Semantic Technology in Business Process Modeling and Analysis. Part 1: Matching, Modeling Support, Correctness and Compliance. EMISA Forum, 15–31.
8. Fellmann, M., Hogrebe, F., Thomas, O., & Nüttgens, M. (2011). Checking the semantic correctness of process models. Enterprise Modelling and Information Systems Architectures, 6(3), 25-35.
9. Fellmann, M., & Zasada, A. (2014). State-of-the-art of business process compliance approaches. In Proceedings of the 22nd European Conference on Information Systems (pp. 1-12).
10. Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R., & Ouenzar, M. (2010). Comparison of model checking tools for information systems. In International Conference on Formal Engineering Methods (pp. 581-596).

11. Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In ICLP/SLP (pp. 1070-1080).
12. Gómez, S. A., Chesnevar, C. I., & Simari, G. R. (2009). Integration of rules and ontologies with defeasible logic programming. In XV Congreso Argentino de Ciencias de la Computación.
13. Governatori, G., Hoffmann, J., Sadiq, S., & Weber, I. (2008). Detecting regulatory compliance for business process models through semantic annotations. In International Conference on Business Process Management (pp. 5-17).
14. Governatori, G., & Rotolo, A. (2010). Norm compliance in business process modeling. In International Workshop on Rules and Rule Markup Languages for the Semantic Web (pp. 194-209).
15. Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description logic programs: combining logic programs with description logic. In Proceedings of the 12th international conference on World Wide Web (pp. 48-57).
16. Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing?. International journal of human-computer studies, 43(5), 907-928.
17. Gruhn, V., & Laue, R. (2007). Checking Properties of Business Process Models with Logic Programming. In MSVVEIS (pp. 84-93).
18. Hitzler, P., Krötzsch, M., Rudolph, S., & Sure, Y. (2007). Semantic Web: Grundlagen. Springer-Verlag.
19. Kontopoulos, E., Bassiliades, N., Governatori, G., & Antoniou, G. (2011). A modal defeasible reasoner of deontic logic for the semantic web. International Journal on Semantic Web and Information Systems (IJSWIS), 7(1), 18-43.
20. Koschmider, A., & Oberweis, A. (2005). Ontology Based Business Process Description. In EMOI-INTEROP.
21. Nielen, A., Költer, D., Mütze-Niewöhner, S., Karla, J., & Schlick, C. M. (2011). An empirical analysis of human performance and error in process model development. In International Conference on Conceptual Modeling (pp. 514-523).
22. Sadiq, S., Governatori, G., & Namiri, K. (2007). Modeling control objectives for business process compliance. In International conference on business process management (pp. 149-164).
23. Shen, Y. D., & Wang, K. (2011). Extending logic programs with description logic expressions for the semantic web. In International Semantic Web Conference (pp. 633-648).
24. Smolnik, S., Teuteberg, F., & Thomas, O. (2012). Semantic Technologies for Business and Information Systems Engineering: Concepts and Applications. Business Science Reference.
25. Thomas, O., & Fellmann, M. (2006). Semantische Integration von Ontologien und Ereignisgesteuerten Prozessketten. In EPK (Vol. 5, pp. 7-23).
26. Van der Aalst, W. M. (1999). Formalization and verification of event-driven process chains. Information and Software technology, 41(10), 639-650.
27. Van der Aalst, W. M., Ramezani, E., & Fahland, D. (2012). Where did i misbehave? diagnostic information in compliance checking. In International conference on business process management (pp. 262-278). Springer Berlin Heidelberg.