Modeling and Checking Business Process Compliance Rules in the Financial Sector

Completed Research Paper

Jörg Becker
University of Münster
European Research Center
for Information Systems (ERCIS)
Leonardo-Campus 3
48149 Münster, Germany
becker@ercis.uni-muenster.de

Philipp Bergener
University of Münster
European Research Center
for Information Systems (ERCIS)
Leonardo-Campus 3
48149 Münster, Germany
bergener@ercis.uni-muenster.de

Patrick Delfmann
University of Münster
European Research Center
for Information Systems (ERCIS)
Leonardo-Campus 3
48149 Münster, Germany
delfmann@ercis.uni-muenster.de

Burkhard Weiß
University of Münster
European Research Center
for Information Systems (ERCIS)
Leonardo-Campus 3
48149 Münster, Germany
weiss@ercis.uni-muenster.de

Abstract

Assuring compliance of business processes with legal and internal regulations is crucial for financial institutions, as non-compliance may lead to severe financial and juridical penalties. To ensure business process compliance, process models have been established as a widely accepted basis for the design, documentation and control of the implementation of business process rules. Accordingly, in this paper, we introduce a semi-automatic business process compliance checking approach based on process models and related models. It relies on graph-based pattern matching, which makes it possible in contrast to existing approaches to define and check any possible type of business rule in any possible type of business process model or even other type of model. The approach is embedded in a design science research methodology.

Keywords: *Business Process Compliance, Business Rules, Conceptual Modeling, Model Checking, Pattern Matching, Financial Sector*

Introduction

Assuring compliance – obeying legal or company-specific regulations – is a crucial issue in information systems development, as neglecting these regulations can lead to severe monetary disadvantages or even legal punishment. Hence every information systems development process should incorporate mechanisms that allow for assuring compliance right from the start. In this paper, we focus on assuring compliance in business processes as a vital part of information systems. Particularly, we focus on business process compliance assurance in the financial sector based on conceptual models.

According to a recent empirical study by Abdullah et al. (2010), "the financial sector is the most highly regulated industry." In fact, with the upcoming of the financial crisis since 2007 and the Euro-crisis since 2010, there is a continuing trend towards increased regulation in this already highly regulated industry (Caldwell 2009, Caldwell et al. 2009, Opromolla 2009). In addition to external regulations, banks are also ruled by many internal regulations, which prescribe the ways and means by which the daily operating business and hence business processes are to be implemented.

As banking business tends to be manifold, with a diverse number of services and products offered to customers, business process landscapes in banks are also very complex. Raduescu et al. (2006), for example, report on a financial institution that engaged in a project "involving over 300 [process] modelers, with a [process] model repository of over 1,800 [business process] models, as part of its $190 million investment in business process management initiatives."

Hence, banks are faced with the immense challenge of ensuring that their large business process repository complies with the abundance of existing laws and new regulations. Supportingly, Moormann et al. (2009) remark that the handling of complexity in business processes of financial institutions is a challenging task, in which some of the most important complexity drivers include the high rate of business rule changes and the redundant documentation of business processes and business rules. Ensuring that business processes comply with given regulations (frequently termed as business process compliance management) is thus a difficult and strenuous task that, if not permanently accomplished successfully, has a high risk of resulting in severe financial and juridical penalties. Winkingly, this dilemma has frequently been described by the bon mot: "If you think compliance is expensive try non-compliance."

Obviously, non-compliance in banks is not an option and therefore, the research problem can be framed as the need to implement business process compliance (BPC) in an efficient (i.e., using as minimal resources as possible) and effective manner (i.e., addressing all regulations that apply to the business process landscape). In recent years, this challenge has led to an increasing number of approaches to solve compliance issues within IS research (Abdullah et al. 2009). In particular, research focused on partially automating the task of compliance checking and giving advice and IT support on how to model business process rules and detect their occurrences – or in case of violation their non-occurrence – in business process models. However, as many of these approaches are still in their infancy (Rikhardsson 2006), they only partially solve the dilemma of business process compliance and are hence not suitable for serving as a possible IS standard with regards to compliance modeling and compliance checking. In fact, there is a research gap that results from the restrictions of these approaches, which can be summarized as follows:

- **types of supported compliance rules:** existing approaches focus on modeling and checking simple business process compliance rules in business process models expressing predecessor/successor relations (e.g., Activity A must follow Activity B). However, more complex rules like escalation rules using additional information from models other than process models are neglected by existing approaches. For instance, a credit application that has been approved by an employee has to be checked once again by a second employee who is superordinate of the first one. An according compliance rule has to check a sequence of actions related to different responsible persons in a process model and the disciplinary relation of the responsible persons in an organizational chart. As such complex rules are not less important than simple ones, we argue that a universal compliance checking approach should not abandon these rules, hence support any rule structure in any type of model.
- **supported (process) modeling languages:** existing approaches focus on compliance checking support for one given process modeling language (e.g., Event-driven process Chains (EPC) (Scheer 2000) or the Business Process Modeling Notation (BPMN) (White and Miers 2008)). Due to the fact that the use of business process modeling languages amongst different banks is very heterogeneous

(Becker et al. 2010), we argue that the compliance community would benefit from a more general compliance checking approach that is suitable for any modeling language. In the end, not every financial institute will be willing to change its process modeling language in use only to fit a specialized compliance checking approach. Thus, a universal compliance checking approach should be applicable to models of any modeling language to enable widespread application.

Hence, our research objective within this paper, and thus also our research contribution in closing the research gap and solving the research problem, is to present a business process compliance checking approach suitable for arbitrary modeling languages. It allows for specifying complex business process compliance rules for any modeling language and for identifying the rules' occurrences in the respective models. Thus, we aim at handling information from process models, combined with other types of linked conceptual models, and also being able to create and analyze complex, e.g. interlaced, business process compliance rules in complex, e.g. highly branched and concurrent (process) models. To achieve this objective, we exploit a well-known property of conceptual models. Every conceptual model can be seen as a graph (Diestel 2010) consisting of nodes and edges. As occurrences of compliance rules appear as typical structures in conceptual models, the search for these occurrences can be described as a graph pattern matching problem – regardless of how complex the rules are and to which type of model the rules are to be applied. Hence, we base our approach upon a graph-based representation of the models to be searched in and a graph pattern-based representation of the compliance rules to be searched for. Two compliance patterns describing the same rule for different modeling languages will hence differ in the used node and edge types and in the way they have to obey the syntaxes of the modeling languages. This way, we are able to define and search for compliance rule patterns of and in models of any type. Of course the set of compliance patterns for a company using, for instance, the Business Process Modeling Notation (BPMN) (White and Miers 2008), and organizational charts will look different from that of a company using, for example, Petri Nets (Petri 1962). Summarizing, the approach to be introduced in this paper differs from existing ones inasmuch the specifics of modeling languages are treated as variables – not as constants. At the end, this allows us as well to specify and search for compliance patterns of any complexity.

Due to the fact that the research problem demands for the development of a new or better approach to business process compliance modeling and checking, and hence for the development of an IS artifact, a design science approach is chosen to tackle the research problem at hand. Typically, design science is applied to design methodologies (including process models) and languages, and as the presented approach deals with compliance modeling and checking in the context of process models, this research approach is particularly well-suited for this type of research.

In particular, the design science research methodology (DSRM), according to Peffers et al. (2007), is applied in this paper. The DSRM approach is a 6-step procedure model that suggests to first identify and define a problem as well as motivate its relevance (as done within this introduction), followed by the definition of objectives of a solution (e.g., requirements that a new design artifact should fulfill to be regarded as a better solution for a given problem). It then proposes to design and develop an artifact (e.g., a new approach to business process compliance modeling and checking), as well as to demonstrate its ability to solve a problem in a suitable context. Furthermore, it recommends to evaluate the artifact's utility and to determine if it is in fact a better solution than existing solutions to solve the given problem. Finally, it demands that the research outcomes shall be published in order to make a valid contribution to the existing scientific body of knowledge.

Thus, the remainder of this paper is structured as follows: at first, the status quo of model-driven business process compliance checking approaches is examined to identify research gaps and requirements for an improved approach to business process compliance modeling and checking. Then, a business process compliance modeling and checking approach suitable for arbitrary modeling languages and based on pattern matching is presented. Furthermore, the functionality of the approach is demonstrated on a set of business process compliance rules, using four different modeling languages. We evaluate our approach by presenting an implementation showing the approach's practical applicability, a performance evaluation to prove the approach's satisfactory running time and a survey including professionals from financial institutions to confirm the utility of the approach. Finally, the paper concludes with a discussion of the key findings and research limitations, while summarizing the research contributions and closing with an outlook on future research that can be inspired by the presented work.

Related Work

The notion of business process compliance refers to the implementation and execution of business processes and their conformance with a set of relevant compliance requirements, such as internal directives and laws that govern them (Sadiq et al. 2007, Cabanillas et al. 2010). To ensure that business processes are compliant with certain rules, conceptual models, and in a narrower sense process models, have been established as a widely accepted basis for the design, documentation and control of the implementation of business process rules (Sadiq et al. 2007, vom Brocke et al. 2008, Wand and Weber 2002).

A methodologically consistent design or maintenance of business processes always begins with the design or maintenance of the corresponding business process models. This approach is propagated by process modeling approaches and has been approved in practice (vom Brocke et al. 2008, Wand et al. 2002). Hence, in order to avoid the risk of violating of compliance rules during design or maintenance of business processes, business process models should be created and maintained in a compliant manner right from the start. However, due to the complex nature of many business processes, and correspondingly their process models, as well as the abundance of business compliance rules and last but not least due to the frequent changes of processes (Moormann et al. 2009), manual business process compliance checking is a strenuous task that should be improved through the use of IT-supported methods (Namiri 2008).

As a consequence, researchers have recently begun with the development of approaches to automate compliance checking and formally model certain types of business process compliance rules. These approaches can be classified by the point in time when compliance is checked. The two points in time are (Cabanillas et al. 2010, El Kharbili et al. 2008, Zoet et al. 2009, Rinderle-Ma et al. 2008) **forward compliance checking (FCC)** (before a process is executed) and **backward compliance checking (BCC)** (after a process is executed).

Backward compliance techniques typically focus on the analysis of actual process instances, obtained from process mining logs from workflow management systems. An example is the compliance checking approach presented by Rozinat and van der Aalst (2008), who present a conformance checking technique that compares the behavior of actual process instances to predefined process models and is able to identify instances that do not comply to the model. A further example of a process mining based backward compliance approach is the approach from van der Aalst et al. (2005), who suggest the use of Linear Temporal Logic to verify business compliance rules for sequence, temporal and executing person constraints. The major drawback of backward compliance checking techniques is, however, "that they can neither prevent the occurrence of non-compliant situations nor modify the behavior of the process instance during its execution to solve problems, since they just compare the results of the execution with the expected behavior, once the process execution is over" (Cabanillas et al. 2010).

In contrary, forward compliance checking techniques concentrate on preventing compliance violations, before they are committed or enabling the on-the-fly recovery of compliance violations directly at the time of their occurrence. This is done by either **design time compliance checking (DTCC)** (checking processes while they are modeled, in terms of a process model), or **run time compliance checking (RTCC)** (checking processes when they are being executed).

As a prerequisite, run time compliance checking techniques need real-time data from running process instances (e.g., from a workflow management system). With these data they can then check if a compliance violation has just taken place and can offer recovery actions to resolve detected compliance violations by restoring a compliant state of the running process instance. An example of such an approach is the semantic mirror approach by Namiri and Stojanovic (2007).

As both backward compliance and run time compliance only provide methods for ex post detection of compliance violations and at best also reactive techniques (in the case of run time compliance checking), research has primarily focused on design time compliance checking techniques (Cabanillas et al. 2010) and resulted in a number of different approaches for these. The approach by Wörzberger et al. (2008) suggests a business process compliance language (BPCL) for the description of compliance rules for inclusion, existence and precedence of activities in processes. BPCL rules are related with WS-BPEL models through an integrated meta model. The business process specification language (BPSL) by Liu et al. (2007) describes another approach for explicating sequence compliance rules for the formal business process execution language (BPEL) that define in which order activities of a process can be executed. Sadiq et al. (2007) describe an approach that furthermore allows for the specification of controls, which can define data and resources needed for a process activity as well as temporal constraints. The paper focuses more on the specification of compliance rules and not on compliance checking, which would only be feasible at runtime for some of the rules. Governatori et al. (2009) present an approach which is also based specifying compliance rules in FCL (Formal Contract Language). They develop an algorithm that is able to detect regulatory compliance violations based on effect annotations in process models. Ghose et al. (2007) propose a similar approach based on effect annotations and compliance rules in CTL (Computational Tree Logic). They focus not so much on compliance checking as such but on strategies to transform incompliant process models to compliant ones.

Under the notion of lifetime compliance of business processes, Ly et al. (2009) suggest a formal framework for DTCC, RTCC and to a minor extent BCC. The framework is based on event traces as a formalization of the underlying processes. Compliance requirements, which cannot be checked during process model creation, are checked during process execution against the current event history, while also considering the future behavior of the process. Special attention is also given to handling process changes.

Comparing this portfolio of research approaches with regards to the utility each approach provides in the context of business rules modeling and compliance checking, we see two main areas of interest regarding a research gap:

- **types of supported compliance rules:** first of all, it is of interest to evaluate the expressiveness of these approaches in terms of being able to model any type of business rule (Cabanillas et al. 2010).
- **supported (process) modeling languages:** secondly, it is of interest to evaluate the universality of the approaches in terms of being able to apply them in the context of any given (process) model and modeling language.

It is important to enable banks to model and check as many process-related compliance rules as possible. However, the approaches by Ghose et al. (2007), Governatori et al. (2009) and Ly et al. (2009), which are based on effect or event notions are all not able to deal with more complex process models containing loops. Ly et al. also have concerns regarding the performance of event trace based checks and instead revert to structural checks on the underlying process models for the prototypical implementation. Liu et al. (2009) do not directly cover the topic of loops. But since their approach is based on Finite State Machines it is unlikely that the approach can deal with the possible infinite number of states caused by loops. Only the approach of Wörzberger et al. (2008) seems to be able to cope with this problem, however it is not clarified. Besides process models, banks, just like many other organizations, also create other types of models, for example, to document their organizational hierarchy and their resources and link them to process models. To enable a maximum freedom in specifying compliance rules, it is also reasonable to use and check information from non-process models (e.g., data and resource models) that are closely linked to process models (cf. also example in the introduction). However, only Sadiq et al. (2007) provide extensive capabilities to define constraints regarding resources and data but do not present an approach to check the constraints. The approaches by Ly et al. (2009) and Governatori et al. (2009) contain variable concepts, which may be able to support process annotations to some extent. However, it is not demonstrated how this can be achieved. Ghose et al. (2007) only take actors into account through the use of BPMN swimlanes. Liu et al. (2007) and Wörzberger et al. (2008) provide no means for any kind of resource related rules. Furthermore, none of the approaches is able to consider separate models for resources.

Supporting any (process) modeling language is of high importance for many organizations including banks from an economic point of view, as many of them have already modeled a large share of their process landscape, while at the same time using individual process modeling languages (Becker et al. 2010). Hence, as it may imply monetary disadvantages and process management "cultural" obstacles, they may not be willing to transform their process models into a language, required by an existing compliance checking approach. Moreover, a company may apply a domain-specific (process) modeling language that fits best its needs. Such a company should be supported by a compliance checking approach anyhow. However, existing approaches are mostly specially tailored for distinct modeling languages. For instance, the approach by Ghose et al. (2007) is tailored towards BPMN, while Liu et al. use BPEL for process modeling. Wörzberger et al. (2009) rely on a meta model integration with WS-BPEL. Governatori et al. (2009) require languages which have a token concept like Petri Nets. Sadiq et al. (2007) state that their approach can be mapped to languages like BPMN and Petri Nets, but do not provide further details or restrictions. Through the use of event traces, the approach of Ly et al. (2009) generally abstracts from specific modeling languages. However, in their prototypical implementation, they rely on structural model checks in ADEPT process models due to performance reasons.

Thus, to our best knowledge, there is no compliance checking approach allowing to define arbitrary compliance rules, based on any type of conceptual model, and suitable for any type of modeling language. Hence, in this paper, we aim to present an approach, which addresses this research gap. This approach shall have the following characteristics:

- **Support any type of compliance rule:** the new approach shall allow the definition of a variety of different types of business process compliance rules, ranging from simple sequence rules, which can also be detected in process models containing loops, to even providing the possibility to specify complex rules that also take advantage of additional information from other conceptual models.
- **Support any modeling language:** the new approach shall support the modeling and checking of compliance rules for arbitrary modeling languages.

Hence, we introduce a business process compliance checking approach that is applicable to conceptual models, regardless of which modeling language they are. Furthermore, we aim at being able to specify compliance rule patterns of any structure – as soon as they are explicable as a subgraph or a subgraph's subdivision in a model graph. This means that regardless of the modeling language a company uses to describe its business processes the approach can be applied without transforming the models of that company into another modeling language. However, the compliance rule patterns have to be specified according to the modeling language the company uses. In contrast to existing approaches, we treat the specifics of a distinct modeling language as variables – not as constants. Consequently, a compliance pattern for BPMN will be different from a pattern for Petri Nets.

A Graph Pattern-based Compliance Checking Approach

According to the previously identified characteristics, a business process compliance checking approach should provide the possibility to specify compliance rules, regardless of their shape, their complexity and the modeling language they are applied to. We argue that this is possible due to two core properties of (process) models and model sections representing compliance rule instances:

- Every (process) model consists of labeled vertices and edges connecting the vertices. In this context, it is irrelevant, which modeling language was used to create the model. Solely the vertices' and edges' types of different modeling languages and the way they can be connected are different. As already stated above, we treat these special properties of modeling languages as variables.
- Every (process) model that obeys a compliance rule incorporates a section that represents the instance of the rule. For example, a compliance rule requiring a document double check will appear as two succeeding process activities with different persons in charge attached. Hence, a compliance rule can be expressed as a (process) model pattern with a distinctive structure and distinctive labeling.

Regarding these two properties, we can specify compliance rules as graph patterns. As it is possible to specify graph patterns of any size, any complexity and any labeling, it is consequently possible to specify any compliance rule using graph patterns. Using graph patterns as a specification basis makes it also possible to handle arbitrary modeling languages.

In order to identify compliance rule instances in (process) models, we make use of a graph pattern matching approach, which was available from a previous research project (Delfmann et al. 2010). It is applicable to multiple modeling languages and multiple application scenarios. Hence, it supports any situation requiring the search for (compliance) patterns in conceptual (process) models.

The idea of this approach is to apply set operations to a set of model elements, representing the model to be analyzed. Coming from graph theory, the approach recognizes any conceptual model as a graph G, consisting of vertices V and edges E, where $G=(V,E)$ with $E \subseteq V \times V$. Therefore, the approach distinguishes model objects, representing nodes, and model relationships, representing edges interrelating model objects. Starting from a basic set that contains all model elements, the approach searches for pattern matches by performing set operations on this basic set. By combining different set operations, patterns are assembled successively. Given a pattern definition, the matching process returns a set of model subsets, representing the pattern matches found. Every match found is put into a separate subset.

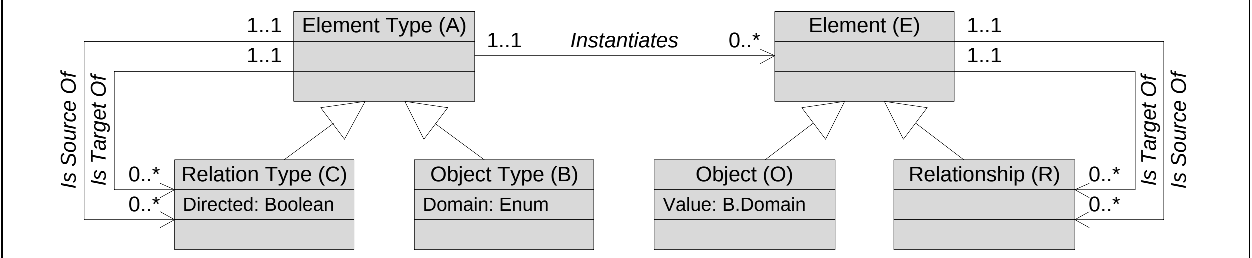

Figure 1. Specification Environment for Conceptual Modeling Languages and Models

As a basis for the definition of patterns, the approach makes use of a specification environment for conceptual modeling languages and models. The specification mainly consists of three constructs (cf. Figure 1). *Element types*, representing any atomic part of a model, are specialized as *object types* (i.e., model vertices) and *relationship types* (e.g., model edges and links). Each relationship type has a *source* element type from which it originates, and a *target* element type to which it leads. Relationship types are either *directed* or *undirected*. Whenever the attribute *directed* is *FALSE*, the direction of the relationship type is ignored. n-ary relationship types are represented as object types connected to n relationship types.

Particular model *elements* are instantiated from their distinct element type. They are specialized as *objects* and *relationships*. Each of the latter leads from a *source element* to a *target element*. Objects can carry *values*, which belong to a distinct *domain*, specified in the object type, to which the object belongs to. For example, the value of an object "name" contains the string of the name (e.g., "credit"). As a consequence, the domain of the object's object type has to be "string" in this case. Thus, attributes are considered as objects.

The pattern matching approach makes use of set operations, extracting elements, objects and relationships with particular properties from the sets of the specification environment shown and thus builds up pattern matches successively. For example, such an operation could analyze all elements of a process model and returns only those process activities that are related to two different organizational units at the same time. This exemplary pattern would represent a compliance rule that requires the so-called four-eyes-principle.

In the following, we introduce the available operations of the approach briefly (for a detailed formal specification cf. (Delfmann et al. 2010)). Each operation has a defined number of input sets and returns a resulting set, where the initial input sets used come from the specification environment (cf. abbreviations in the objects of Figure 1). In the explanation of the operations, we use additional sets (*X*: arbitrary set of elements; *Y*: arbitrary set of objects; *Z*: arbitrary set of relationships), specifying which types of inputs an operation expects. The first category of operations reveals specific properties of model elements:

- *ElementsOfType(X,a)* returns a set of all elements of *X*, belonging to the given element type *a*.
- *ObjectsWithValue(Y,value)* returns a set of all objects of *Y*, whose *values* equal the given one.
- *ObjectsWithDomain(Y,domain)* returns a set of all objects of *Y*, whose *domains* equal the given one.

In order to assemble complex pattern structures successively, the following operations combine elements and their relationships and elements, being related, respectively:

- *ElementsWithRelations(X,Z)* returns a set of sets, containing all elements of *X* and their undirected relationships of *Z*. Each inner set contains one occurrence.
- *ElementsWithOutRelations(X,Z)* returns a set of sets, containing all elements of *X* and their directed, outgoing relationships of *Z*. Each inner set contains one occurrence.
- *ElementsWithInRelations(X,Z)* is defined analogously to *ElementsWithOutRelations*. In contrast, it only returns incoming relationships.
- *ElementsDirectlyRelated (X_1,X_2)* returns a set of sets, containing all elements of X_1 and X_2 that are connected directly via undirected relationships of *R*, including these relationships. Each inner set contains one occurrence.
- *DirectSuccessors (X_1,X_2)* is defined similarly to *ElementsDirectlyRelated*. Though, it only returns directed relationships, of which the source elements are part of X_1 and the target elements are part of X_2.

A further category of operations is needed to build patterns, representing recursive structures (e.g. a path of an arbitrary length):

- *{Directed}Paths(X_1,X_n)* returns a set of sets, containing all sequences with undirected {directed} relationships, leading from any element of X_1 to any element of X_n. The elements that are part of the paths do not necessarily have to be elements of X_1 or X_n, but can also be of $E\backslash X_1\backslash X_n$. Each path found is represented by an inner set.
- *{Directed}Loops(X)* is defined similarly to *{Directed} Paths*. It returns a set of sets, containing all undirected {directed} sequences, which lead from any element of *X* to itself.

To avoid infinite sets, only finite paths and loops are returned. To provide a convenient specification environment for structural model patterns, we define some additional functions that are derived from those already introduced:

- *ElementsWith{In|Out}RelationsOfType(X,Z,c)* returns a set of sets, containing all elements of *X* and their {un}directed, {incoming|outgoing} relationships of *Z* of the type *c*. Each occurrence is represented by an inner set.
- *ElementsWithNumberOf{In|Out}Relations(X,n)* returns a set of sets, containing all elements of *X*, which are connected to the given number *n* of {un}directed {incoming|outgoing} relationships of *R*, including these relationships. Each occurrence is represented by an inner set.
- *ElementsWithNumberOf{In|Out}RelationsOfType(X,c,n)* returns a set of sets, containing all elements of *X*, which are connected to the given number *n* of {un}directed {incoming|outgoing} relationships of *R* of the type *c*, including these relationships. Each occurrence is represented by an inner set.
- *{Directed}PathsContainingElements(X_1,X_n,X_c)* returns a set of sets, containing elements that represent all undirected {directed} paths from elements of X_1 to elements of X_n, which each contain at least one element of X_c. The elements that are part of the paths do not necessarily have to be elements of X_1 or X_n, but can also be of $E\backslash X_1\backslash X_n$. Each such path found is represented by an inner set.
- *{Directed}PathsNotContainingElements(X_1,X_n,X_c)* is defined similarly to *{Directed}PathsContaining-Elements*. However, it only returns paths that do not contain any element of X_c.
- *{Directed}Loops{Not}ContainingElements(X,X_c)* is defined similarly to *{Directed}Paths{Not}ContainingElements* but is related to loops.

By nesting the functions, introduced above, it is possible to build model patterns successively. The results of each function can be reused, adopting them as an input for other functions. In order to combine different results, the basic set operators *union* ($\cup$), *intersection* ($\cap$), and *complement* ($\backslash$) can generally be used. Since it should be possible to not only combine sets of pattern matches (i.e., sets of sets), but also the pattern matches themselves, the approach incorporates additional set operators. These operate on the inner sets of two sets of sets respectively.

The *Join* operator performs a *union* operation on each inner set of the first set with each inner set of the second set. Since we regard patterns as cohesive, only inner sets that have at least one element in common, are considered. The *InnerIntersection (II)* operator *intersects* each inner set of the first set with each inner set of the second set. The *InnerComplement (IC)* operator applies a *complement* operation to each inner set of the first outer set combined with each inner set of the second outer set. Only inner sets that have at least one element in common are considered.

As most of the introduced set operations expect simple sets of elements as inputs, further operators are introduced that turn sets of sets into simple sets. The *SelfUnion (SU)* operator merges all inner sets of one set of sets into a single set performing a *union* operation on all inner sets. The *SelfIntersection (SI)* operator performs an *intersection* operation on all inner sets of a set of sets successively. The result is a set, containing elements that each occur in all inner sets of the original outer set.

A simple exemplary pattern, searching for two particular Activities (named “Activity A” and “Activity B”), following each other over a path of arbitrary length, is specified as follows:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Activity),"Activity A"),
              ObjectsWithValues(ElementsOfType(O,Activity),"Activity B"))≠∅
```

As this pattern shows, some compliance rules require the analysis of model vertices’ labels. To receive consistent results, it is crucial that the labels are semantically unambiguous. Although we do not evaluate according approaches in this paper, we recommend using an approach that is able to consider the actual *meaning* of a model element's content. Otherwise proper pattern matching is hardly possible, as simply checking the strings of labels may lead to ambiguous, incorrect or even no results. To account for using according approaches, we use placeholders in the patterns to be specified below that are supposed to carry semantic concepts to be evaluated by the respective approaches (see below).

Application to Different Modeling Languages and Compliance Rules

In the following we show how the pattern matching approach can be applied to models of different modeling languages and to different compliance checking scenarios. For this purpose we use a set of compliance rules that were identified through literature reviews in prior research (Becker et al. 2011). Although, this set of rules does not claim to be exhaustive, it covers a bulk of compliances rules that are discussed in the literature. Hence, they should provide a good measure of the applicability of the pattern matching approach for compliance modeling and checking.

As process modeling language examples, we use BPMN, Event-driven Process Chains (EPCs) (Scheer 2000) and Petri Nets (PN). As a non-process modeling language example, we use organizational charts, which play an important role in compliance management, for example, due to escalation regulations. To show the applicability of our compliance checking approach to different process modeling languages, we specify typical business process compliance rules for process modeling languages, which we took from the literature (Awad and Weske 2009). In addition, to show the applicability to other conceptual models, we specify a compliance rule requiring information from both process models and organizational charts.

Table 1. Object Types of Process Modeling Languages

BPMN	EPC	Petri Nets
Activity	Function	Transition
{Start|Intermediate|End}Event	Event	Place
Gateway	Operator	---
Document	Data	Place
IT System	IT System	Place
Swimlane	Organizational Unit	Place
…	…	…

Table 2. Relationship Types of Process Modeling Languages

Language	Relationship Type	Source Object Type	Target Object Type	Directed
BPMN	Process Flow 1	Activity	Event	TRUE
BPMN	Process Flow 2	Event	Activity	TRUE
BPMN	Process Flow 3	Activity	Gateway	TRUE
BPMN	Responsibility	Activity	Swimlane	FALSE
…	…	…	…	…
EPC	Control Flow 1	Function	Event	TRUE
EPC	Control Flow 2	Event	Function	TRUE
EPC	Control Flow 3	Function	Operator	TRUE
EPC	Responsibility	Function	Organizational Unit	FALSE
…	…	…	…	…
Petri Nets	Flow 1	Transition	Place	TRUE
Petri Nets	Flow 2	Place	Transition	TRUE
Petri Nets	Marking	Place	Token	FALSE

Tables 1 and 2 exemplarily show how process modeling languages can be specified using the specification environment introduced above. Their object types are listed in Table 1. Due to the fact that the process modeling languages we use here are all designed to represent actions and incidents, they all incorporate similar concepts. (Note that there are further process modeling languages that do not make use of incident-like concepts and rely on actions only. Their process models will of course differ from those designed in one of the languages at hand. The same applies for according compliance patterns). For instance, actions are called "Activities" in BPMN, "Functions" in EPCs and "Transitions" in Petri Nets. Incidents are called "Events" in BPMN and EPCs and are represented through "Places" in Petri Nets. There exist equal concepts for annotations in BPMN and EPCs, such as Documents/Data, IT-Systems and Organization. In Petri Nets, such annotations are represented through special places (e.g., a place representing an organizational unit, whose tokens are consumed and reproduced as soon as the according transition fires). According relationship types are listed in Table 2. Just like the object types, the relationship types of the process modeling languages show many similarities. The Control/Process Flow is always a directed relationship type between an action and an incident. In BPMN and EPCs, annotations are connected to activities/functions through undirected relationship types to indicate their use (note that some dialects use directed relationship types, e.g., for data inputs and outputs). Petri Nets do not incorporate annotations explicitly, so they simply use the common control flow to connect special places to transitions (cf. example above). Since BPMN and EPCs are very similar in their languages, their compliance patterns will look quite similar, whilst a Petri Net pattern for the same purpose will look slightly different.

Table 3. Element Types of Organizational Charts

Object Type (OT)
Organizational Unit (OU)
Job
Person
…

Relationship Type	Source OT	Target OT	Directed
part_of	OU	OU	TRUE
belongs_to	OU	OU	TRUE
supervises	Job	Job	TRUE
occupies	Person	Job	TRUE
Fayol's Bridge	Person	Person	FALSE
…	…	…	…

Table 3 shows the specification of a typical modeling language of organizational charts. It consists of Organizational units (OU), jobs and persons that are interrelated by *part-of* relationships (e.g. to express that a department belongs to a division), *belongs-to* relationships to assign jobs to OUs, *supervises* relationships to express who is superordinate to whom, etc.

Applying the pattern matching approach to business process compliance checking requires identifying typical business rules representing requirements by law or internal standards. In the following, we introduce exemplary patterns representing such structures. We apply them to the different modeling languages specified above to provide an impression of the general applicability of our approach. The patterns can be used to check (process) models whether they contain the specified structures and, as a consequence, whether they comply with the given rules.

To allow a convenient specification of patterns, which are similar for the introduced process modeling languages, we make use of placeholders comprising similar object types of the languages in the following. `Action` will be a placeholder for `Activity`, `Function` and `Transition`. `Incident` will comprise `Event` and `Place`. Of course `Transition` is only to be used with `Place`, `Function` with `Event`, and so on. Accordingly, we generally speak of *Actions* and *Incidents* in the following.

In particular, we define patterns expressing so-called *control flow rules*, *data rules* and *resource rules*, according to Awad and Weske (2009). Control flow rules define the sequence in which actions may or should be performed. As general concepts, we introduce predecessor relations (if there is a particular action, then it has to be preceded by another particular action) and successor relations (if there is a particular action, then it has to be followed by another particular action). Furthermore, there are existence or non-existence rules (e.g., a process has to contain a particular action). Resource and data rules focus annotations of process actions. This means that not only the action sequences are taken into account, but also their relationships to organization responsible for their execution and data being processed during execution. Control flow patterns to be specified are *(1) mandatory action*, *(2) forbidden action*, *(3) mandatory predecessor*, *(4) forbidden predecessor*, *(5) mandatory successor*, *(6) forbidden successor*, *(7) mandatory intermediary*, *(8) forbidden intermediary*, *(9) start action*, *(10) end action*, and *(11) immediate succession*. Resource and data patterns are *(12) separation of duties*, *(13) four-eyes-principle*, and *(14) effect sequencing*. A sophisticated resource rule that we specify to demonstrate parallel compliance checking in models of different modeling languages is *(15) escalation*.

Note that in any case where a model element's label has to be checked, meaning that a pattern searches for a model element with a specific content, we use placeholders like `ACT_A` ("Action A") supposed to contain terms, phrases, ontological concepts, etc. We use these placeholders to indicate that checking the contents of a model element against a given value can be done by using an arbitrary approach – for example by using the *Levenshtein distance* (Levenshtein 1966), the *ontological similarity* (e.g., Thomas and Fellmann 2009) or the *linguistic similarity* (e.g., Delfmann et al. 2009). In the first case, `ACT_A=ACT_B` means that the Levenshtein distance of `ACT_A` and `ACT_B` equals zero, in the second case `ACT_A` and `ACT_B` are ontologically equal, and in the third case the terms (or even the phrases) used in the element's labels are synonymous. Although we do not evaluate according approaches in this paper, we recommend using an approach that is able to consider the actual *meaning* of a model element's content. Otherwise a proper comparison is hardly possible.

The *mandatory action* rule requires a particular action to be part of a process. That is, a compliance check based on the pattern matching approach introduced has to search for that action (i.e., the returned element set of the following pattern specification must not be empty):

```
ObjectsWithValues(ElementsOfType(O,Action),ACT_A)≠∅
```

The *forbidden action* rule requires a particular action not to be part of a process. Thus, the pattern matching approach has to search the according model for such an action. Here, the returned element set of the pattern specification *must* be empty):

```
ObjectsWithValues(ElementsOfType(O,Action),ACT_A)=∅
```

The *mandatory predecessor* rule requires an action to precede another action, as soon as the latter exists. In particular, this means that there has to exist a path in the process model that leads from the preceding action over an arbitrary number of other actions to the succeeding action. First, it has to be checked whether a specific action (here: `ACT_B`) exists:

```
ObjectsWithValues(ElementsOfType(O,Action),ACT_B)≠∅
```

If `ACT_B` exists, then a path from a specific predecessor action (here: `ACT_A)` to `ACT_B` has to exist in order to satisfy the predecessor constraint:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_B))≠∅
```

Should the second pattern search return no result, then a compliance violation is detected. If `ACT_B` does not exist, the second check is not necessary, as then it is not possible to violate a predecessor constraint related to `ACT_B`.

The *forbidden predecessor* rule requires an action not to precede another action, as soon as the latter exists. In particular, this means that a path in the process model that leads from the preceding action over an arbitrary number of other actions to the succeeding action must not exist. First, it has to be checked whether `ACT_B` exists:

```
ObjectsWithValues(ElementsOfType(O,Action),ACT_B)≠∅
```

If `ACT_B` exists, then a path from `ACT_A` to `ACT_B` *must not* exist in order to satisfy the predecessor constraint (i.e., the returned element set of the following pattern specification *must* be empty):

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_B))=∅
```

Should the second pattern search return a non-empty set, then a compliance violation is detected. If `ACT_B` does not exist, the second check is not necessary, as then it is not possible to violate a predecessor constraint related to `ACT_B`.

The *mandatory* and *forbidden successor* rules represent the counterparts of the *mandatory successor* and *forbidden successor* ones. Here, it is required/forbidden that a particular action is *followed* by another one. Thus, the specifications of the compliance rules are analogous, with `ACT_A` and `ACT_B` exchanged.

The *mandatory intermediary* rule requires that as soon as two particular succeeding actions exist, they have to be intermediated by a third particular action. First, it has to be checked whether `ACT_A` exists and is followed by `ACT_C`:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_C))=∅
```

If such a path exists, then this path has to contain `ACT_B`:

```
DirectedPathsContainingElements(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
   ObjectsWithValues(ElementsOfType(O,Action),ACT_C),
   ObjectsWithValues(ElementsOfType(O,Action),ACT_B))≠∅
```

Should the second pattern search return no result, then a compliance violation is detected. If there is no path from `ACT_A` to `ACT_C`, the second check is not necessary, as then it is not possible to violate a mandatory intermediary constraint related to `ACT_A` and `ACT_C`.

The *forbidden intermediary* rule requires that as soon as two particular succeeding actions exist, they must not be intermediated by a third particular action. First, it has to be checked whether `ACT_A` exists and is followed by `ACT_C`:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_C))=∅
```

If such a path exists, then this path must not contain `ACT_B`:

```
DirectedPathsNotContainingElements(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
   ObjectsWithValues(ElementsOfType(O,Action),ACT_C),
   ObjectsWithValues(ElementsOfType(O,Action),ACT_B))≠∅
```

Should the second pattern search return no result, then a compliance violation is detected. If there is no path from `ACT_A` to `ACT_C`, the second check is not necessary, as then it is not possible to violate a forbidden intermediary constraint related to `ACT_A` and `ACT_C`.

The *start action* rule requires a particular action to be the start action of a process. Accordingly, such an action (here: `ACT_A`) has to be preceded by a starting incident. The BPMN pattern checks whether `ACT_A` is preceded by a start event. The EPC pattern checks whether there is an event having no predecessors, which precedes `ACT_A`. A Petri Net pattern has to check whether `ACT_A` is preceded by a place that has no predecessors. Furthermore, `ACT_A` must not have any other predecessors:

```
BPMN:  DirectSuccessors(ElementsOfType(O,StartEvent),
          ObjectsWithValues(ElementsOfType(O,Activity),ACT_A))≠∅

EPC:   DirectSuccessors(SU ElementsWithNumberOfInRelations(ElementsOfType(O,Event),0),
          ObjectsWithValues(ElementsOfType(O,Function),ACT_A))≠∅

PN:    DirectSuccessors(SU ElementsWithNumberOfInRelations(ElementsOfType(O,Place),0),
          ObjectsWithValues(SU ElementsWithNumberOfInRelations(
             ElementsOfType(O,Transition),1),ACT_A))≠∅
```

The *end action* rule is the counterpart of the *start action* one. It requires a particular action to be the end action of a process. Accordingly, such an action (here: `ACT_A`) has to be succeeded by an end incident. The BPMN pattern checks whether `ACT_A` is succeeded by an end event. The EPC pattern checks whether there is an event having no successors, which succeeds `ACT_A`. A Petri Net pattern has to check whether `ACT_A` is succeeded by a place that has no successors. Furthermore, `ACT_A` must not have any further successors:

```
BPMN:  DirectSuccessors(ObjectsWithValues(ElementsOfType(O,Activity),ACT_A),
          ElementsOfType(O,EndEvent))≠∅

EPC:   DirectSuccessors(ObjectsWithValues(ElementsOfType(O,Function),ACT_A),
          SU ElementsWithNumberOfOutRelations(ElementsOfType(O,Event),0)))≠∅

PN:    DirectSuccessors(ObjectsWithValues(
          SU ElementsWithNumberOfOutRelations(ElementsOfType(O,Transition),1),ACT_A)
          SU ElementsWithNumberOfOutRelations(ElementsOfType(O,Place),0)))≠∅
```

The *immediate succession* rule requires two actions to succeed each other directly without any other action intermediating them. First, it has to be checked whether there exists `ACT_A` that is followed by `ACT_B`:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_B))=∅
```

If such succeeding actions exist, we have to assure that they are not intermediated by one or more further actions. Thus, they have to be direct successors:

```
DirectSuccessors(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
                 ObjectsWithValues(ElementsOfType(O,Action),ACT_B))=∅
```

The *separation of duties* rule requires two succeeding actions to be executed by different responsible persons. First, it has to be checked whether `ACT_A` and `ACT_B` exist succeeding each other:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Action),ACT_B))≠∅
```

If such succeeding actions exist, then they have to be assigned to different persons, or more generally speaking, organizational units. The pattern for BPMN checks whether the two activities are assigned to different swimlanes. The EPC pattern returns succeeding functions that are annotated by different organizational units (OU). The Petri Net pattern checks whether the succeeding transitions have different input places serving as organizational units:

```
BPMN/EPC: DirectedPaths(ElementsOfType(O,Action) II ElementsDirectlyRelated(
              ObjectsWithValues(ElementsOfType(O,Action),ACT_A),
              ObjectsWithValues(ElementsOfType(O,OU|Swimlane),Org1)),
            ElementsOfType(O,Action) II ElementsDirectlyRelated(
              ObjectsWithValues(ElementsOfType(O,Action),ACT_B),
              ObjectsWithValues(ElementsOfType(O,OU|Swimlane),Org2)))≠∅;    Org1≠Org2

PN:       DirectedPaths(ElementsOfType(O,Transition) II DirectSuccessors(
              ObjectsWithValues(ElementsOfType(O,Transition),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Place),Org1)),
            ElementsOfType(O,Transition) II DirectSuccessors(
              ObjectsWithValues(ElementsOfType(O,Transition),ACT_B),
              ObjectsWithValues(ElementsOfType(O,Place),Org2)))≠∅;          Org1≠Org2
```

A resource rule that is similar to separation of duties is the *four-eyes-principle*. It requires a single action to be executed by two different persons. First, it has to be checked whether `ACT_A` exists:

```
ObjectsWithValues(ElementsOfType(O,Action),ACT_A)≠∅
```

If such an action exists, it has to be checked whether the activity is executed by two persons. In case of BPMN, the pattern checks if the same activity can be found on two different swimlanes (in case of BPMN, the activity has to be duplicated, as in BPMN it is not possible to assign one single activity to two swimlanes at the same time). The EPC pattern checks whether the function is assigned to two different organizational units. The Petri Net pattern returns transitions that have two different input places serving both as organizational units:

```
BPMN: ElementsDirectlyRelated(ObjectsWithValues(ElementsOfType(O,Activity),ACT_A),
          ObjectsWithValues(ElementsOfType(O,Swimlane),Org1)) UNION
      ElementsDirectlyRelated(ObjectsWithValues(ElementsOfType(O,Activity),ACT_A),
          ObjectsWithValues(ElementsOfType(O,Swimlane),Org2))≠∅;            Org1≠Org2

EPC:  ElementsDirectlyRelated(ObjectsWithValues(ElementsOfType(O,Function),ACT_A),
          ObjectsWithValues(ElementsOfType(O,OU),Org1)) JOIN
      ElementsDirectlyRelated(ObjectsWithValues(ElementsOfType(O,Function),ACT_A),
          ObjectsWithValues(ElementsOfType(O,OU),Org2))≠∅;                  Org1≠Org2

PN:   DirectSuccessors(ObjectsWithValues(ElementsOfType(O,Transition),ACT_A),
          ObjectsWithValues(ElementsOfType(O,Place),Org1)) JOIN
      DirectSuccessors(ObjectsWithValues(ElementsOfType(O,Transition),ACT_A),
          ObjectsWithValues(ElementsOfType(O,Place),Org2))≠∅;               Org1≠Org2
```

Following Zoet et al. (2009), rules for *effect sequencing* describe business objects with certain properties implying further actions to be executed (e.g. credit applicants applying for credits worth more than

75,000 $ must receive an additional positive vote inside a bank). For example, a compliance rule requiring the assignment of business objects with specific properties to specific actions is specified as follows: First, it has to be checked whether a business object exists that is related to an attribute, whose value describes a specific property. In BPMN and EPC models, we search for Document/Data objects, which carry an according attribute. Petri Nets are searched for places carrying an according token:

```
BPMN/EPC: ElementsDirectlyRelated(ElementsOfType(O,Document|Data),
            ObjectsWithValues(ElementsOfType(O,Attribute),Constraint))≠∅

PN:       ElementsDirectlyRelated(ElementsOfType(O,Place),
            ObjectsWithValues(ElementsOfType(O,Token),Constraint))≠∅
```

As the property could be anything, we indicate this property by the term “constraint” in the example. If there is a business object having this property, it has to be checked if it has been assigned to the required action – in this case ACT_A. In BPMN and EPC models, this is done by annotation. In Petri Nets, the place carrying the business object token has to be an input place of ACT_A:

```
BPMN/EPC: ElementsDirectlyRelated(ElementsOfType(O,Document|Data) II
            ElementsDirectlyRelated(ElementsOfType(O,Document|Data),
              ObjectsWithValues(ElementsOfType(O,Attribute),Constraint)),
            ObjectsWithValues(ElementsOfType(O,Action),ACT_A))≠∅

PN:       DirectSuccessors(ElementsOfType(O,Place) II
            ElementsDirectlyRelated(ElementsOfType(O,Place),
              ObjectsWithValues(ElementsOfType(O,Token),Constraint)),
            ObjectsWithValues(ElementsOfType(O,Transition),ACT_A))≠∅
```

The possibility to build patterns for arbitrary compliance rules and arbitrary languages results from the universal applicability of the underlying pattern matching approach. It does not require modeling language-specific types of objects or relationships. The specifics of an according modeling language are not part of the pattern matching approach, but they are specified as variables during the specification of the compliance patterns as shown above. Of course, a pattern specified for one modeling language cannot be applied to another one, as in such a case a transformation mechanism would be necessary. Nevertheless, the approach can be applied by any company, regardless of which (process) modeling language it uses.

To illustrate the applicability of the approach to other modeling languages and even process models combined with other types of models, we specify an advanced *escalation* compliance rule. It requires subsequent checks of a transaction by employees of different disciplinary positions. For example, a credit application that has been approved by an employee has to be checked once again by a second employee superordinate of the first one. An according compliance rule has to check a sequence of actions related to different responsible persons in a process model and the disciplinary relation of the responsible persons in an organizational chart. The following example illustrates this rule with EPCs and organizational charts:

First, it has to be checked whether two functions representing both check actions (here: ACT_A and ACT_B) exist succeeding each other:

```
DirectedPaths(ObjectsWithValues(ElementsOfType(O,Function),ACT_A),
              ObjectsWithValues(ElementsOfType(O,Function),ACT_B))≠∅
```

If such succeeding functions exist, then they have to be assigned to different jobs. Moreover, the second job has to be superordinate of the first one:

```
DirectedPaths(ElementsOfType(O,Function) II ElementsDirectlyRelated(
      ObjectsWithValues(ElementsOfType(O,Function),ACT_A),
      ObjectsWithValues(ElementsOfType(O,Job),Job1) II
        DirectedPathsNotContainingElements(
          ObjectsWithValues(ElementsOfType(O,Job),Job2),
          ObjectsWithValues(ElementsOfType(O,Job),Job1),
          R \ ElementsOfType(R,supervises))),
   ElementsOfType(O,Function) II ElementsDirectlyRelated(
      ObjectsWithValues(ElementsOfType(O,Function),ACT_B),
      ObjectsWithValues(ElementsOfType(O,Job),Job2) II
        DirectedPathsNotContainingElements(
          ObjectsWithValues(ElementsOfType(O,Job),Job2),
          ObjectsWithValues(ElementsOfType(O,Job),Job1),
          R \ ElementsOfType(R,supervises))))≠∅;                   Job1≠Job2
```

To realize more sophisticated compliance rules, two or more of the rules shown can be combined.

Applicability, Performance and Utility Evaluation

To demonstrate the value of our compliance checking approach, it is necessary to prove its applicability, its performance and its utility.

In order to show the approach's **applicability**, we have implemented a plug-in for a meta modeling tool that was available from a former research project (Delfmann et al. 2009). The tool consists of a meta modeling environment that is based on the specification environment for modeling languages shown in Figure 1.

The plug-in provides a specification interface for compliance patterns. It is integrated into the meta modeling environment of the tool, as the patterns have to be specified according to the respective modeling language as shown above. All basic sets, functions and set operators introduced previously are provided and can be used to build up structural model patterns successively. In order to gain a better overview of the patterns, they are displayed and edited in a tree-like structure. Users can build up the tree-structure through drag-and-drop of the basic sets, functions and set operators.

The patterns specified can be applied to any model that is available within the model base and that was developed with the modeling language, fitting the pattern specification. Applying a pattern reveals all of its occurrences existing in the model base. The occurrences are displayed through highlighting their model nodes and edges. To gain a quick overview, the models containing the pattern occurrences can be flipped.

As the general pattern matching problem in graphs has an exponential computational complexity, we had to prove that the **performance** of the compliance checking approach is satisfactory. For this purpose, we conducted an empirical performance evaluation (a detailed explanation can be found in Dietrich et al. 2011). We analyzed models of different modeling languages having a size ranging from 20 to 343 elements. We used patterns of different size and different density (i.e. containing more or less arcs per node). The patterns ranged from simple exact subgraphs to those containing paths of arbitrary length. For the performance test, we used a common personal computer equipped with an Intel® Core™ 2 Duo CPU E8400 3.0 GHZ with 3.25 GB RAM and Windows 7 (32-Bit edition). The result shows that the pattern matching approach returns results with acceptable performance. Most of the patterns could be identified within a few milliseconds.

Concerning the overhead produced by the specification of the compliance rule patterns, we bring forward two arguments that militate in favor of our approach: Firstly, we determined the average time needed by a trained person to specify a pattern. We measured a few minutes for a typical compliance rule. This means that even specifying hundreds of patterns with the introduction of compliance management using our approach only takes a few days. New regulations that may become relevant day-to-day can be added quickly. The problem that remains is to identify the compliance rules relevant for the according company. However, this problem remains for any automated or manual compliance approach as it is highly individual.

To estimate the real **utility** of our approach, we conducted a case study in the banking sector. We selected a regional bank, which was operating in one of the major cities and its surrounding metropolitan areas across two federal states in Germany. As a universal bank, it offered its clients typical products such as consumer credits, investment counseling, credit cards, and giro accounts with many additional services and products offered for special client groups such as corporate clients, startup founders, and students. It was serving more than 632,000 customers with more than 2,200 employees in over 130 branch offices, with a balance sheet total of 9.9 billion Euros in 2009. Being part of the cooperative banking sector in Germany, it was among the top 10 largest banks in Germany.

We selected its bank account opening process for new customers, since this process is one of the most universal and frequent processes that can be found in almost any commercial bank. In addition, we received further compliance related material that the bank's legal department supplied for this special process (e.g., applicable internal and external regulations). Through several interviews and observations at the bank, this process was modeled using BPMN and evaluated as valid by the bank representatives. Overall, the process had to obey 12 business process compliance rules (most of them due to internal regulations). We specified the compliance rules as patterns and applied them to the process model. As a result, we found that it was possible to specify each business rule using our business process compliance modeling notation and it was also possible to automatically detect the compliance of the business processes with these business rules, using our prototypical modeling tool implementation. As already our performance evaluation suggests, the performance of the compliance checking process was satisfactory.

In a workshop with bank representatives responsible for process management and compliance, we presented the process model, the relevant compliance rules, the way how to specify the rules and the way how to check compliance of the process model. Furthermore, we presented a complete compliance checking task in order to provide an impression of our approach's performance. The approach was rated highly beneficial by all of the bank's process management and compliance representatives. In particular, they appreciated the applicability to different modeling languages. This was due to special attributes the bank is using to annotate process model activities. Furthermore, the bank was planning to extend their modeling language by law-related annotations. The speed of checking multiple compliance rules in one step rather than searching for them manually was appreciated as well. The bank representatives were aware of the fact that selecting the relevant compliance rules and specifying them in a tool is time-consuming. However, they regarded this task as not avoidable. We repeated the workshop with representatives of another German bank acting in the field of pharmacy and a provider of IT services for a large cooperative society of banks. The feedback of the according representatives was equal.

Discussion, Limitations and Outlook

In this paper, we have introduced a model-driven business process compliance checking approach that is suitable for models of any modeling language and for business process compliance rules of any structure. The approach exploits a common property of conceptual models, in particular their representation as graphs. As occurrences of compliance rules appear as typical structures in conceptual models, the search for these occurrences can be described as a graph pattern matching problem, which we addressed by introducing a graph pattern matching based business process compliance checking approach. The approach differs from existing approaches as it is not restricted to a special process modeling language and as it is not restricted to simple linear compliance rules. To prove its applicability to multiple modeling languages, we provided a set of common compliance rules and specified them for different modeling languages, including a non-process modeling language. The evaluation of our approach consisted of an implementation as a modeling prototype, a performance evaluation and a survey confirming its utility.

The applicability to any modeling language and any compliance rule closes the research gap identified in the introduction and the related work section. The contributions of our approach to practice and research and their implications are manifold:

- Companies that use a modeling language different from those used in language-specific approaches are provided with a semi-automatic business process compliance checking approach. Before, those companies were either forced to change their modeling language, which causes not only a time-consuming transformation process but also may cause acceptance and skill problems, or they had to perform manual compliance checking. Hence, the approach implies an increase of the ability of semi-automatic compliance checking for a number of companies and is therefore a valuable contribution for practice.
- The presented approach allows for checking compliance rules of any structure. Previous approaches were restricted to linear compliance rules. This means that compliance rules, which had to be checked manually before, due to previously existing restrictions, can now be checked automatically (or at least semi-automatically). As this implies an increase of efficiency in business process compliance management, we identify another valuable contribution for practice.
- The introduced approach generalizes the identification of linear compliance rule occurrences in process models of a distinct modeling language towards the identification of arbitrary compliance rule occurrences in any conceptual model. Hence, the approach extends the previously limited scope of business process compliance checking to a generalized level and contributes to the existing literature body of knowledge in business process compliance checking, discussed in the related work section. As we

have shown that our approach works for different languages and different compliance patterns, we argue that a restriction to a special modeling language and only simple compliance rules, as proposed by the existing approaches, discussed in the related work section, is not necessary.

- As the approach is not restricted in its set of rules, it implies to be applied to related purposes requiring pattern matching in conceptual models, like for example model comparison, model integration, business process weakness detection, model transformation, and syntax checking. Therefore, it contributes indirectly to the literature body of knowledge in these research fields. Hence, we plan to apply our approach to model comparison, model integration, business process weakness detection, model transformation, and syntax checking in the future.

Despite the contributions, our presented approach delivers, there are also some limitations:

- With regards to automation and therefore efficiency, our approach still requires the identification of relevant compliance rules (e.g., from law or company-internal regulatory documents) and their transformation into formal compliance rule patterns. This task is time-consuming; however, we doubt that it is automatable. Furthermore, the initial specification of the relevant set of compliance patterns has to take place only once. The patterns can be reused repeatedly for compliance checks whenever new (process) models have been created.
- Another issue of automation is the fact that compliance checking cannot be automated completely by our approach; again we doubt that this is possible at all. To our experience, gained from discussions with professionals, a great amount of time is consumed by searching processes or process models for those sections that require compliance or that may violate a compliance rule. Our approach is only suitable to reduce this searching time and find as many compliance violations as possible. This can be of high value for a compliance manager, as it makes the annoying and time-consuming search process obsolete. Nevertheless, the final decision, if a possible compliance violation has to lead to a process change, is made by the compliance manager. Hence, our approach does not aim at replacing the expertise of a compliance manager.
- The approach is applicable to multiple modeling languages. However, as already stated, once a compliance rule is defined for one modeling language, it will not work on another one (cf. the differences of compliance rules for different languages in the application section). Companies using different modeling languages will not be able to exchange their compliance rules.

Although the performance evaluation has shown that there are no performance problems, we still aim to improve the performance, making the approach also work for very large model repositories, containing models with thousands of nodes. This can become relevant when regarding the overall model of a company that is not split up into model parts (e.g., the whole process landscape of a company). In this context, our future research will address efficient algorithms from graph theory, which exploit the fact that conceptual models are typically sparse graphs.

In this paper, we did not go into detail on the importance of standardizing natural language elements that are often used to specify elements of (process) models (e.g. activities, resources, organizational units etc.). However, as already stated, this is an important aspect, as business process compliance rules can only be specified well, if also a common vocabulary is used. Here, we have only referred to prior research describing different approaches to standardizing semantics of model element labels, for instance by providing linguistic conventions or connecting the models to ontologies. However, we are aware of the fact that generally accepted or standardized labeling is a crucial issue in business process modeling. Otherwise, models may become ambiguous and identifying a model element with a particular meaning as part of a compliance rule occurrence becomes nearly impossible. Therefore, we aim at combining our previous work on semantic disambiguation of conceptual models (Delfmann et al. 2009) with the presented approach.

With regards to user acceptance and utility, we have only conducted a preliminary evaluation. Although the professionals that were involved in the evaluation confirmed a high value of the approach for business process compliance checking, we aim at extending the utility evaluation by performing extensive tests in a real-world compliance management environment. At the very moment, we are applying the approach in the institutions that were also involved in the preliminary evaluation. In particular, the evaluation comprises modeling of the relevant processes, specification of relevant compliance rules, executing the checking approach, measuring the increase/decrease of detected compliance violations compared to manual checking, measuring time savings and determining possible options for usability improvement.

References

Abdullah, S. N., Indulska, M., and Sadiq, S. 2009. "A Study of Compliance Management in Information Systems Research." in *Proceedings of the 17th European Conference on Information Systems (ECIS 2009)*, Verona, Italy, pp. 1-10.

Abdullah, S. N., Sadiq, S., and Indulska, M. 2010. "Emerging Challenges in Information Systems Research for Regulatory Compliance Management," *Lecture Notes in Computer Science* (6051), pp. 251-265.

Awad, A., and Weske, M. 2009. "Visualization of Compliance Violation in Business Process Models," in *Proceedings of the 5th Workshop on Business Process Intelligence*, pp. 1-12.

Becker, J., Bergener, P., Delfmann, P., Eggert, M., and Weiß, B. 2011. „Supporting Business Process Compliance in Financial Institutions – A Model-Driven Approach," in *Proceedings of the 10th International Conference on Wirtschaftsinformatik*. Zurich, 16.-18. February.

Becker, J., Breuker, D., Weiß, B., and Winkelmann, A. 2010. „Exploring the Status Quo of Business Process Modelling Languages in the Banking Sector – An Empirical Insight into the Usage of Methods in Banks," in *Proceedings of the 21st Australasian Conference on Information Systems (ACIS 2010)*, Brisbane, Australia.

Cabanillas, C., Resinas, M., and Ruiz-Cortés, A. "Hints on how to face business process compliance," *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos* (4:4), pp. 26-32.

Caldwell, F. 2009. *The Worldwide Economic Crisis will Bring Real-Time Reporting for Risk Management*. Gartner Research, Gartner, Inc., Stamford.

Caldwell, F., Bace, J., and Lotto, R. J. D. 2009. *U.S. Financial System Regulatory Overhaul Brings More Scrutiny*. Gartner Research. Gartner, Inc., Stamford.

Delfmann, P., Herwig, S., and Lis, L. 2009. "Unified Enterprise Knowledge Representation with Conceptual Models - Capturing Corporate Language in Naming Conventions," In *Proceedings of the 30th International Conference on Information Systems (ICIS 2009)*. Phoenix, Arizona, USA.

Delfmann, P., Herwig, S., Lis, L., Stein, A., Tent, K., and Becker, J. 2010 "Pattern Specification and Matching in Conceptual Models. A Generic Approach Based on Set Operations," *Enterprise Modelling and Information Systems Architectures* (5:3), pp. 24-43.

Diestel, R. 2010. *"Graph Theory"* 4th Edition, Heidelberg, Germany.

Dietrich, H., Steinhorst, M., Becker, J., and Delfmann, P. 2011. „Fast Pattern Matching in Conceptual Models – Evaluating and Extending a Generic Approach," appears in *Proceedings of the 4th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2011)*, Hamburg, Germany.

El Kharbili, M., de Medeiros, A., Stein, S., and van der Aalst, W. M. P. 2008. "Business Process Compliance Checking: Current State and Future Challenges," *Lecture Notes in Informatics* (141), pp. 107-113.

Ghose, A.K., and Koliadis, G. 2007. "Auditing business process compliance," in *Proceedings of the International Conference on Service-Oriented Computing (ICSOC-2007)*, pp. 169-180.

Governatori, G., Hoffmann, J., Sadiq, S., and Weber, I. 2009. "Detecting Regulatory Compliance for Business Process Models through Semantic Annotations," *Lecture Notes in Business Information Processing* (7:1), pp. 5-17.

Levenshtein, V. 1966. "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Doklady* (10), pp. 707–10.

Liu, X., Müller, S., and Xu, K. 2007 "A static compliance-checking framework for business process models," *IBM Systems Journal* (46:2), pp. 335-361.

Ly, L., Rinderle-Ma, S., Göser, K., and Dadam, P. 2009 "On enabling integrated process compliance with semantic constraints in process management systems: Requirements, challenges, solutions," *Information Systems Frontiers* (accepted for publication).

Moormann, J., Vetter, D., and Hilgert, M. 2009. „Reducing Complexity: Business Rules in Business Process Management," (in German), *Die Bank* (109:11), 2009, pp. 30-37.

Namiri, K. 2008. *"Model-Driven Management of Internal Controls for Business Process Compliance"*, Doctoral Thesis, University of Karlsruhe.

Namiri, K., and Stojanovic, N. 2007. "A Formal Approach for Internal Controls Compliance in Business Processes," in *Proceedings of the 8th Workshop on Business Process Modeling, Development and Support (BPMDS 2007)*, Trondheim, Norway, pp. 1-9.

Opromolla, G. 2009. “Facing the Financial Crisis: Bank of Italy's Implementing Regulation on Hedge Funds.” *Journal of Investment Compliance* (10: 2), pp. 41-44.

Peffers, K., Tuuanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems* (24:3), pp. 45-77.

Petri, C. A. 1962. *“Communicating with Machines [in German: Kommunikation mit Automaten]“*. Ph. D. Thesis. University of Bonn.

Raduescu, C., Tan, H. M., Jayaganesh, M., Bandara, W., zur Muehlen, M., and Lippe, S. 2006. "A Framework of Issues in Large Process Modeling Projects,“ in *Proceedings of the European Conference on Information Systems (ECIS 2006)*, Göteborg, Sweden, pp. 1-12.

Rikhardsson, P., Best, P., Green, P., and Rosemann, M. 2006. “Business Process Risk Man-agement, Compliance and Internal Control: A Research Agenda,” in *Proceedings of the 2nd Asia/Pacific Research Symposium on Accounting Information Systems*, Melbourne, Australia.

Rinderle-Ma, S., Ly, L. T., Dadam, P. 2008. „Business Process Compliance,“ *EMISA Forum* (28:2), pp. 24-29.

Rozinat, A., and van der Aalst, W. M. P. 2008. "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems* (33:1), pp. 64-95.

Sadiq, S., Governatori, G., and Namiri, K. 2007. “Modeling control objectives for business process compliance,” in *Proceedings of the 5th International Conference on Business Process Management (BPM 2007)*, pp. 149-164.

Scheer, A.-W. 2000. *“ARIS – Business Process Modelling”*. 3rd Edition. Berlin, Germany.

Thomas, O., and Fellmann, M. 2009. “Semantic Process Modeling – Design and Implementation of an Ontology-Based Representation of Business Processes,” *Business & Information Systems Engineering (1:6)*, 438-451.

van der Aalst, W. M. P., de Beer, H. T., and van Dongen, B. F. 2005. "Process Mining and Verification of Properties: An Approach Based on Temporal Logic," *Lecture Notes in Computer Science* (3760), pp. 130-147.

vom Brocke, J., Becker, J., Simons, A., and Fleischer, S. 2008. "Conceptual Modeling of Enterprise Content," in *Proceedings of the 7th International Conference on Perspectives in Business Informatics Research (BIR 2008)*, Gdańsk, Poland.

Wand, Y., and Weber, R. 2002. "Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda," *Information Systems Research* (13:4), pp. 363-376.

White, S. A., and Miers, D. 2008. “*BPMN Modeling and Reference Guide. Understanding and Using BPMN,*” Future Strategies Inc., Lighthouse Point, FL, USA.

Wörzberger, R., Kurpick, T., and Heer, T. 2008. "Checking Correctness and Compliance of Integrated Process Models," in *Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scienti*fic Computing, pp. 576-583.

Zoet, M., Welke, R., Versendaal, J., Ravesteyn, P. 2009. „Aligning Risk Management and Compliance Considerations with Business Process Development,” *Lecture Notes in Computer Science* (5692), pp. 157-168.