Checking Business Process Models for Compliance – Comparing Graph Matching and Temporal Logic

Dennis M. Riehle

University of Münster, European Research Center for Information Systems, Münster, Germany
dennis.riehle@ercis.uni-muenster.de

Abstract. Business Process Compliance Management (BPCM) is an integral part of Business Process Management (BPM). A key objective of BPCM is to ensure and maintain compliance of business processes models with certain regulations, e.g. governmental laws. As legislation may change fast and unexpectedly, automated techniques for compliance checking are of great interest among researchers and practitioners. Two dominant concepts in this area are graph-based pattern matching and pattern matching based on temporal logic. This paper compares these two approaches by implementing four compliance patterns from literature with both approaches. It discusses what requirements both approaches have towards business process models and shows how to meet them. The results show that temporal logic is not able to fully capture all four patterns.

Keywords: Process Querying, Pattern Matching, Compliance Patterns, Graph Matching, GMQL, Temporal Logic, CTL.

1 Compliance of Business Process Models

With the rise of Business Process Management (BPM) over the last decades, organizations have created large amounts of conceptual models [1], making conceptual models an integral and important part of modern organizations. One important aspect of BPM is Business Process Compliance Management (BPCM). BPCM is about checking if business processes adhere to agreed-upon objectives, to gain and to maintain such compliance [2]. Objectives which define a compliant process may come from several sources, as internal and external requirements. Though organizations may specify internal rules and objectives which business processes need to comply with, many compliance requirements come from legislation (e.g. [3, 4]). As governmental regulations may change frequently and at unexpected times, BPCM must react to such external changes in a fast, dynamic and efficient way. These requirements towards BPCM call for tools, techniques and methodologies, to support BPCM, by finding compliance violations in an automatic and simple manner [5].

One approach towards automatic process analysis is Business Process Querying [6], where a set of (structural) patterns is queried against a repository of process models (e.g. [4]). This requires a query language, where a pattern is used as an input and queried against an arbitrary amount of models (the repository), to check whether these models

contain the searched pattern, and is seen as an enabler business intelligence [7]. In literature, there are plenty of query languages that can be used to identify compliance violations in process models (e.g. [8–11]). Query languages can be divided into structural and behavioural query languages [12], where behavioural query languages tend to operate directly on process logs, while structural query languages tend to operate on a process model. Two well-known but methodically different concepts are graph matching for structural querying and temporal logic for both structural and behavioural querying. While authors of either approaches describe the advantages of their approach (see [8–11] for details), the research goal of this paper is to clarify which are the advantages and disadvantages of using either graph matching or temporal logic. Hereby, this paper only focuses on model-based compliance checking, where process models are analysed for compliance. Four compliance patterns from literature are specified using both approaches – as far as possible – and the results are compared.

This paper is structured as follows. In Section 2 shortly discusses modelling query languages in general, while in Section 3, the transformation of business models to structures suitable for analysis is discussed. Section 4 presents the specification of different queries using the techniques of graph matching and temporal logic. Section 5 compares graph matching and temporal logic and Section 6 concludes with an outlook.

2 Background and Related Work

Business Process Management (BPM) is about the "concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes" ([13], p. 5). One key part of BPM is the representation of business processes through business process models, for which a variety of modelling languages exists (see, e.g., [14] or [15] for an overview).

One core concern in the analysis of business processes is the identification of structural and behavioural patterns in process models. In literature, there is a variety of model query languages, which enable the specification of such patterns and provide a technique to identify occurrences of them in process models (e.g., [16]). One common concept is graph matching, which tries to match a pattern (a small graph) within a model (a large graph). When found, this small graph would be called a match and represent part of the whole model (a subgraph that is either equal or similar to the pattern). Consequently, if graph matching does not return any results, the pattern was not found in the model. Contrastingly, temporal logic formulates rules and assertions, which are evaluated against a formalized version of a process model. These rules and assertions are then processed and result in a binary decision, true or false. The contextual interpretation of this result of course depends on the way the query was formulated.

2.1 Compliance Checking with Graph Matching

One exemplary model querying language based on graph matching is the Generic Model Query Language (GMQL), which provides pre-defined sets of elements and a batch of functions to apply to these elements [11]. Functions in GMQL can be nested

to any level, to provide a maximum of flexibility. Functions to identify elements include *ElementsOfType (EOT)* and *ElementsWithAttributeOfValue (EWAOV)*, which do what their name suggests and otherwise are defined in [11]. Functions to identify elements including their related neighbours include *ElementsWith{NumberOf}{Pred|Succ} Relations{Of-Type} (EW{N}{P|S}R{OT}, ElementsDirectlyRelated (EDR)* and *Adjacent-Successors (AS)*. Additionally, there are functions to identify paths and loops within the model, including *{Directed}Paths{[Not]ContainingElements} ({D}P{{N}CE})* and *{Directed}Loops{[Not]Containing-Elements} ({D}L{{N}CE})*. Further operators exist, which can combine two result sets into one set. These operators include *{Self}Union*, *Join*, *{Inner|Self}Intersection* and *{Inner}Complement*. All of these functions are defined in the relevant literature.

GMQL has already been applied to the scenario of compliance checking, where compliance patterns have been derived from legislation and where these patterns have been searched in real-world process models [3].

2.2 Compliance Checking with Temporal Logic

Another way of model querying is the Computational Tree Logic (CTL). CTL originates from the specification and verification of software systems. An introduction to model checking with CTL can, for instance, be found in [17]. CTL includes several statements, which describe states. The operator X can be used to describe the next state; the operator F describes any state in the future. To describe all states, G can be used (global) and U can be used describe all states until a certain condition appears to be true. Additionally, the operators X, F and G can be preceded by an A, which requires all paths to fulfil the criteria, or with E, which requires that there is at least one path matching the criteria. The exclamation mark can be used to negate conditions.

Linear Temporal Logic (LTL) and CTL have already been used for compliance checking of process models. For example, [4] and [18] have specified compliance patterns using temporal logic. Additionally, [4] specifies a Compliance Request Language (CRL) as a set of abbreviations for commonly used CTL/LTL expressions. This makes the specification of compliance patterns shorter and easier to understand.

3 Transforming Process Models to State-Machines

Graph matching and temporal logic pose different demands on the business process models they shall be applied to. While the pattern is specified with GMQL on the one hand and CTL, LTL or CRL on the other hand, there are different requirements towards the model as well. Graph matching requires the model to be in a graph structure, that is, a structure consisting of vertices, edges, and attributes, and temporal logic requires the process models to be in a formal representation like a computation tree that can be processed with CTL. Conceptual models like business process models consist of elements and relations between these elements, hence they fulfil the general criteria of a graph structure. However, depending on the Business Process Modelling Language (BPML) used, process models may not be computation trees and, hence, require further

pre-processing and transformations, before temporal logic can be applied. For demonstration purposes, business processes modelled with a domain-specific BPML (icebricks, [19]) are used. Since domain-specific BPMLs usually follow similar concepts of frameworks (e.g., [20]), other BPMLs are likely to face similar challenges during the transformation. There are two important characteristics of icebricks: First, main processes and detail processes can consist of multiple main process variants or detail process variants. These variants are used to reflect different or partly different activities performed in different instances of the same business process. Such a variant might reflect the same business process, like opening a bank account, for either a private customer or a business customer. Since one process instance only reflects one variant, different variants can be compared to XOR splits in other BPMLs. icebricks models do not have directed edges, as models are – by convention – read from top to bottom.

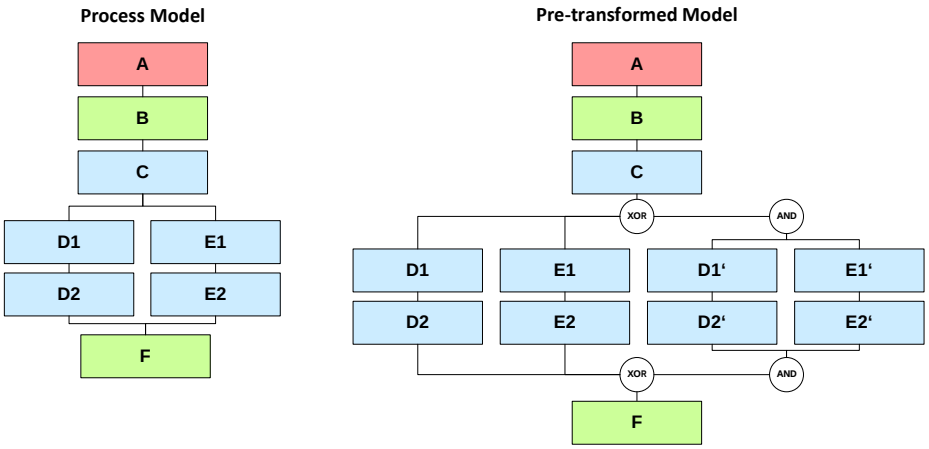

Figure 1. Original and pre-transformed process model.

Second, several process elements can succeed one process element and one process element can be preceded by multiple elements (see process model in Figure 1). For this modelling scenario, icebricks does not define which of the process branches needs to be executed, as long as at least one branch appears in a process instance. Consequently, they are to be considered as an inclusive OR. While one may argue that this unclear execution semantic is a design flaw, the originators of icebricks argue that this simplification enables the creation of process models which are easy to understand, even for non-technicians. Additionally, other popular process modelling languages like Event-driven Process Chains [21] do include a similar OR connector and whose semantics have been discussed thoroughly (e.g., [22]). As a model check with CTL requires a formally specified process model as input, the execution semantics of the icebricks models needs to be clarified first. This can be easily done by replacing OR constructs through a combination of XOR and AND.

The example given in Figure 1 shows a simple process. It consists of one main process (A), one detail process (B) and five process activities. The process activities D1 and E1 follow the process activity C, which means that after executing C, either only

D1 and D2 can be executed, or only E1 and E2 can be executed, or all of them (D1, D2, E1 and E2) can be executed. To keep the computation tree simple, it is assumed that if multiple process branches are executed, each process branch is fully executed before another process branch starts (this means that there is no real parallel execution). With this assumption in mind, the tree shown in Figure 2 can be created, showing all possible executions paths throughout the process model from Figure 1.

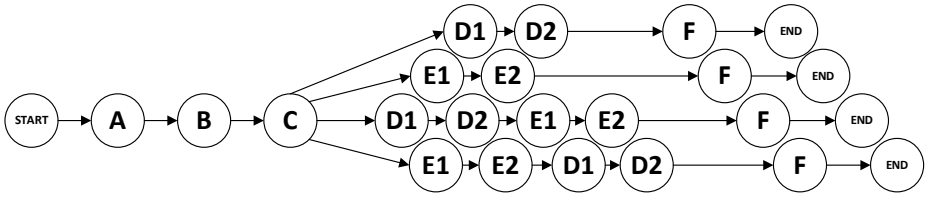

Figure 2. Process model as computation tree.

While the example above with the two process activities D1 and E1 following C is rather simple, the pre-transformation becomes drastically more complex if there are more process branches. In particular, *n* process branches will result in a total amount of *(2^n)-1* combinations when still disregarding the order of execution. If we would further regard the order in which process branches are executed, the term k! would be added as a multiplication inside the sum, which explains why the amount of execution paths regarding the execution order of activities explodes with rising numbers of process branches. This issue is also known as the state explosion problem [23].

For a short demonstration, a process database from a financial service provider has been transformed from icebricks to computation trees using the logic described above. This database contained 31 main processes and 216 detail processes, with 2,894 process activities altogether (Table 1 only shows four sample processes). For the process model before and after expansion, the number of activities (Act.) and gateways (Gate.) in the model is depicted. Obviously, the number of states in a computation tree would be even lager in the end, as many activities are duplicated.

Process Model	**Process Model**		**After Expansion**	
	Act.	Gate.	Act.	Gate.
Current Account	39	4	3,042	1.030
Savings Account	19	4	603	252
Construction Loan	56	2	393	24
Consumer Loan	13	2	29	10

Table 1. Complexity of process models after transformations.

Besides the unclear execution semantics introduced by an inclusive OR operator, BPMLs can also contain other structures which need special attention when process models are to be transformed in computation trees. Probably the most prominent structure to be found in many BPMLs are loops. For demonstration purposes, the commonly known language Business Process Model and Notation (BPMN) [24] is used.

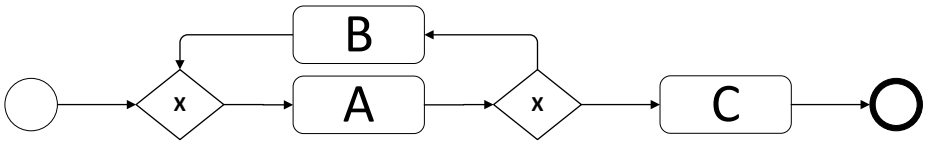

Figure 3. Process model containing a loop.

Figure 3 depicts a sample BPMN process containing a small loop. A computation tree built for such a process model should correctly reflect the execution semantic of that process model. This means that activity B can occur between to executions of A and activity A can occur between two executions of B. Of course, A can also occur without any execution of B. These assumptions also holds true if there were more than just one activity on the loop. In addition, the loop can be executed unlimited times, which makes an adequate representation in a computation tree impossible. As branches in trees only split but never merge, a tree cannot reflect a loop.

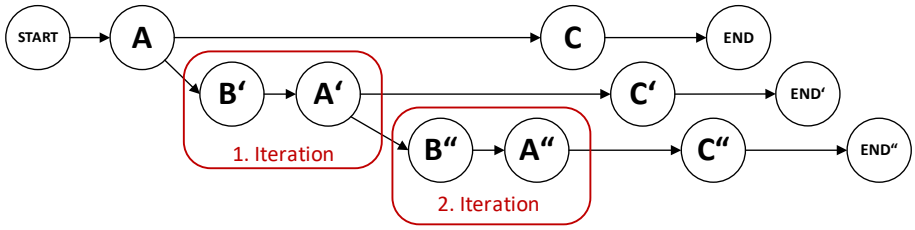

Figure 4. Process model with loop as computation tree.

As a work-around to convert process models containing loops to computation trees, only a limited amount of loop executions is simulated. More precisely, in this paper exactly two loop executions are simulated, as depicted in Figure 4. This is just enough to fulfil the previously mentioned criteria: A can occur for itself, A can occur between two executions of B and B occurs between two executions of A. Again, the problem of state explosion is still present, as all elements following the loop need to be duplicated two times. Therefore, loops still create computational overhead.

4 Exemplary Compliance Patterns

GMQL and CTL provide different features for compliance checking. To enable a comprehensive comparison of both languages, four different compliance patterns are presented and, where possible, specified for both languages. While the first two patterns directly originate from literature, i.e., they were already specified with either GMQL or CTL, the last two patterns are motivated from requirements towards compliance checking approaches or textual descriptions. For these patterns, both the GQML query and the CTL statement have been newly developed.

The first compliance pattern originates from [3] where it was named infringement pattern #1 and was originally specified in GMQL. This compliance pattern derives from

the German securities trading act [25], which states that when banks consult customers in order to sell them new products, they need to hand out all necessary product information to the customer, before the product is sold, meaning before any transactions are made. In this example, such process activities are named “Consult customer”, “Talk to customer”, “Hand out contract”, “Hand out documents”, “Hand out preliminary contract”, “Make account transaction” and “Perform account transaction.”

Name: Infringement pattern	**Reference:** [3]
The pattern searches for a process element, where an advice is given as well as an account transaction made. If no contract is handed out in between the rule is violated. (in accordance to WpHG § 31 (4a))	
GMQL	DPNCE (UNION (EWAOV (O,'Caption','Consult customer'), EWAOV (O,'Caption','Talk to customer')), UNION (EWAOV (O,'Caption','Make account transaction'), EWAOV (O,'Caption',‘Perform account transaction')), UNION (UNION (EWAOV (O, 'Caption', 'Hand out contract'), EWAOV (O, 'Caption', 'Hand out documents')), EWAOV (O, 'Caption', 'Hand out preliminary contract'))))
CTL	M, s0 |= AG ((“Consult customer” ∨ “Talk to customer” → AG (EG (¬“Make account transaction” ∧ ¬ “Perform account transaction”) ∨ A [¬“Make account transaction” ∧ ¬ “Perform account transaction”) U (“Hand out contract" ∨ “Hand out documents” ∨ “Hand out preliminary contract”)))

Table 2. Example Pattern “Infringement pattern”.

The GMQL query shown in Table 2 returns compliance violations. This is done by searching for all paths that start with a process activity named either “Consult customer” or “Talk to customer” and end in an activity named “Make account transaction” or “Perform account transaction”, where the directed path does not contain any element named either “Hand out contract”, “Hand out documents” or “Hand out preliminary contract.” An empty result set does consequently imply that no violations are found.

Name: Too Many Routing Paths per Element	**Reference:** [26]
The pattern refers to situations where a process contains too many different variants starting at the same object. (Three or more variants in this example)	
GMQL	COMPLEMENT (O, UNION (UNION (EWNSR (O, 0), EWNSR (O, 1)), EWNSR (O, 2)))
CTL	Not possible

Table 3. Example Pattern “Too Many Routing Paths per Element”.

A corresponding CTL statement is also shown in Table 2. The statement checks if there is an activity called “Consult customer” or “Talk to customer” somewhere in the process model. If so, for all paths reachable from there must hold that it is not allowed to reach an activity called “Make account transaction” or “Perform account transaction”

before there was not another activity called “Hand out contract" or “Hand out documents” or “Hand out preliminary contract”. If an element “Make account transaction” or “Perform account transaction” was found earlier, the statement evaluates to FALSE and indicates a compliance violation.

The second example originates from [26] and addresses the issue that models may be too complex for the participants to efficiently work with them. The pattern addresses this problem in the context of process models which consist of multiple variants. Such models may become overly complicated due to the occurrence of too many variants at a certain position in the model. In this pattern, the threshold of variants starting at any given element is set to three.

In the corresponding GMQL query (see Table 3) this is done by building the complement of all available elements in a model and the elements of the model which have zero, one or two succeeding relations. As it can be seen for each number of outgoing relations below the threshold two additional statements (union and EWNSR) have to be used. Increasing the threshold therefore leads to a more complex query. In contrast to the GMQL this pattern cannot be built with CTL, as the language does not support the identification of the number of succeeding or preceding elements.

Name: Check documents repetitively		**Reference: -**
The weakness pattern addresses situations where a document is checked more than once in a loop of a process		
GMQL	DLCE (O, UNION (EWAOV (O,'Caption','Check document'), EWAOV (O,'Caption','Analyse document')))	
CTL	M, s0 |= AG ((“Check document” ∨ “Analyse document”) → AG (¬“Check document” ∧ ¬“Analyse document”))	

Table 4. Example Pattern “Check documents repetitively”.

A mandatory requirement of a compliance query language in practice is the identification of complex graph structures as stated in [3], e.g., loops. In the following, two patterns are used to showcase the limitations of the languages regarding such structures. Therefore, the third pattern is used to identify processes in which an activity is conducted multiple times in a process due to a loop although the activity is performed redundantly after the first execution. In this example a document check named “Check document” or “Analyse document” should only be checked once. In the GMQL query, depicted in Table 4, all loops are selected which do contain an element with the name “Check document” or “Analyse document”. In case one or more loops are found, the model violates the pattern.

The CTL statement also shown in Table 4 works differently. It does not explicitly select loops. Instead it is searched if an element called “Check document” or “Analyse document” exists. In case it does, all following paths are not allowed to have any states labelled “Check document” or “Analyse document”. In comparison to the GMQL query the usage of the CTL statement has two weaknesses. Firstly, during the transformation of the pre-transformed process model into a computation tree, the loop has to be passed

through at least two times. If the loop is only traversed once, the loop's states are only included one time in the computation tree and can thus not be detected. Secondly, the CTL statement does not only detect the elements within loops, but also within the whole process. This may lead to false negative results, in which multiple occurrences of the states not being in a loop are detected. The result of the expression would in turn be FALSE, although the pattern is not violated.

Name: Observe iteration efficiency		**Reference: -**
The pattern identifies if the efficiency of a loop is observed each iteration		
GMQL	DLNCE (O, UNION (EWAOV (O, 'Caption', 'Check if exception handling necessary'), EWAOV (O, 'Caption', 'Check for exception handling')))	
CTL	Not possible	

Table 5. Example Pattern "Observe iteration efficiency".

The fourth example pattern is again concerned with complex structures. It states that a loop should always contain an activity "Check if exception handling necessary" or "Check for exception handling", which has the aim to determine if the loop's iteration should be cancelled and an exception handling process should be started. The purpose of the pattern is the identification of possible inefficiencies and delays in a process, caused by loops not being observed closely and interrupted, if to many iterations occur.

As depicted in Table 5 the GMQL query looks for loops which do not contain any of the two activities "Check if exception handling necessary" or "Check for exception handling". The pattern is violated, if the result set is not empty.

Using CTL this pattern cannot be expressed. In contrast to the previous example it is necessary to be able to identify loops, as it is the basic prerequisite implying the other conditions. The creation of a partially correct CTL statement is therefore not possible.

5 Comparison of Graph Matching and Temporal Logic

Graph matching and temporal logic are two different methodologies, which both can be applied in the context of compliance management and which both already have been applied in practice. Because of their different underlying theoretical concepts, the two methodologies provide a different set of functions and operations (cf. section 2.1 and 2.2). As displayed in section 4, the result of this difference is that two of the example patterns are not expressible in CTL and one does not entirely reflect the pattern. In particular, CTL is missing the features to count elements and to detect loops. The latter can in some cases be circumvented by using the transformation procedure described in section 3 and additionally formulating the CTL statement as a path constraint as shown with the pattern "Check documents repetitively" in section 4. Nonetheless, the resulting statement produces only partially correct results and its applicability is practice is hence questionable. These weaknesses of the CTL approach seem to arise from the necessity

to use a computation tree. During the transformation process information gets lost, i.e., loops since they have to be unwound, or blurred, i.e., preceding or succeeding elements cannot be simply detected anymore.

Another disadvantage arising from the transform process is the state explosion problem. As shown in section 3, this transformation can lead to large state machines, which grow exponentially with the amount of branching elements (e.g., OR, XOR) in a process model. In return, GMQL can operate directly on the process model and is thus not affected by this issue. The author further argues that compliance checking need to be applicable in process models with multiple concurrent executed process flows, as process modelling languages such as icebricks and EPC contain branching elements.

Lastly, regarding the visualization of compliance violations, there is a significant difference comparing graph matching with temporal logic. As graph matching is able to directly operate on the process model, the pattern is directly mapped to elements of the process model using algorithms of graph matching. If the pattern is found in the model, there is a relation of elements in the pattern to the matched elements in the model, which allows the GMQL implementation provided by [3] to highlight the matched parts directly in the model. This is possible, because [3] provides an implementation in form of a plugin as part of a modelling tool. The visualization of matched results is a very convenient feature for users, as it makes it easy to find the part of a process which has caused the compliance pattern to match – even for very large process models. Contrastingly, to the best of the author's knowledge, a comparable visualization of results is not possible, as CTL, LTL and CRL only return a Boolean value whether a process contains a certain compliance pattern or not. Furthermore, it is questionable if such a visualization feature could be implemented using CTL, as CTL only operates on the computation trees and therefore, there is no direct relation of identifiers used in the compliance pattern and elements in the process model.

6 Conclusion & Outlook

This paper has shown and discussed different approaches for checking business process models for compliance violations. Computational Tree Logic as a representative of temporal logic was used and compared it to the Generic Model Query Language, representing graph matching. By specifying compliance patterns for both approaches, it was possible to point out a gap in the capabilities of both languages in regards to compliance checking. The examples presented revealed several disadvantages of CTL in comparison to GMQL.

Firstly, two patterns could not expressed using CTL, which shows that CTL is not fully capable to capture the functionality of GMQL. As only four patterns were specified, the author suggests that further research should focus on these edge cases, by comparing the functionalities of GMQL and CTL further. This may reveal situations in which GMQL is not applicable, whereas CTL is. In general, further research in this area would allow to state more accurately the limitations of GMQL and CTL compared to each other, to improve both approaches in the future.

Secondly, the importance of the visualisation of compliance violations was highlighted. As CTL only returns TRUE or FALSE for a given statement, this is not supported by CTL. The author therefore asks researchers to focus on this issue, as an easy-to-use tool with a comprehensible visualization could greatly improve the usage of pattern-based compliance management in practice.

Lastly, there are several compliance patterns mentioned in literature. However, these patterns are widely spread over different papers and usually are only specified for one approach, if not only specified textually. Therefore, no integrated catalogue exists providing an overview of all compliance patterns researchers have described up till today. Future Research should develop such an integrated catalogue of compliance patterns and to invent techniques for automatically transferring compliance patterns, e.g., from CTL to GMQL and vice versa. Additionally, practitioners could benefit from the presence of such a catalogue and researchers could conduct empirical studies on the effectiveness of BPCM in practice with regard to compliance patterns.

References

1. Dijkman, R., Rosa, M.L., Reijers, H. a.: Managing large collections of business process models - Current techniques and challenges. Comput. Ind. 63, 91–97 (2012).
2. Sadiq, S., Governatori, G., Namiri, K.: Modeling Control Objectives for Business Process Compliance. In: Proceedings of the 5th International Conference on Business Process Management (BPM). pp. 149–164. , Brisbane, Australia (2007).
3. Becker, J., Delfmann, P., Dietrich, H.-A., Steinhorst, M., Eggert, M.: Business Process Compliance Checking – Applying and Evaluating a Generic Pattern Matching Approach for Conceptual Models in the Financial Sector. Inf. Syst. Front. 1–47 (2014).
4. Elgammal, A., Turetken, O., van den Heuvel, W.-J., Papazoglou, M.: Formalizing and Appling Compliance Patterns for Business Process Compliance. Softw. Syst. Model. 1–28 (2014).
5. Höhenberger, S., Riehle, D.M., Delfmann, P.: From Legislation to potential Compliance Violations in Business Processes-Simplicity Matters. In: ECIS. p. ResearchPaper188 (2016).
6. Polyvyanyy, A.: Business Process Querying. In: Sakr, S. and Zomaya, A. (eds.) Encyclopedia of Big Data Technologies. Springer, Cham (2018).
7. Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decis. Support Syst. 100, 41–56 (2017).
8. Awad: BPMN-Q: A Language to Query Business Processes. In: Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures. pp. 115–128. , St. Goar, Germany (2007).
9. Störrle, H.: VMQL: A visual language for ad-hoc model querying. J. Vis. Lang. Comput. 22, 3–29 (2011).

10. Ouyang, C., Hofstede, A.T.: APQL: a process-model query language. In: Proceedings of the 1st Asia Pacific Conference on Business Process Modelling (AP-BPM2013). pp. 23–38. , Beijing, China (2013).
11. Delfmann, P., Steinhorst, M., Dietrich, H.-A., Becker, J.: The Generic Model Query Language GMQL – Conceptual Specification, Implementation, and Runtime Evaluation. Inf. Syst. 47, 129–177 (2015).
12. Deutch, D., Milo, T.: A structural/temporal query language for Business Processes. J. Comput. Syst. Sci. 78, 583–609 (2012).
13. Weske, M.: Business Process Management: Concepts, Methods, Technology. Springer, Berlin (2007).
14. List, B., Korherr, B.: An Evaluation of Conceptual Business Process Modelling Languages. In: Proceedings of the 2006 ACM symposium on Applied computing. pp. 1532–1539. , Dijon, France (2006).
15. Lu, R., Sadiq, S.: A survey of comparative business process modeling approaches. In: Proceedings of the 10th International Conference on Business Information Systems (BIS2007). pp. 82–94. , Poznan, Poland (2007).
16. Becker, J., Delfmann, P., Eggert, M., Schwittay, S.: Generalizability and Applicability of Model-Based Business Process Compliance-Checking Approaches - A State-of-the-Art Analysis and Research Roadmap. BuR - Bus. Res. 5, 221–247 (2012).
17. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge, USA (2000).
18. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using BPMN-Q and temporal logic. Lect. Notes Comput. Sci. Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinforma. 5240 LNCS, 326–341 (2008).
19. Becker, J., Clever, N., Holler, J., Shitkova, M.: Icebricks - Business Process Modeling on the Basis of Semantic Standardization. In: International Conference on Design Science Research in Information Systems. pp. 394–399. Springer (2013).
20. Jannaber, S., Riehle, D.M., Delfmann, P., Thomas, O., Becker, J.: Designing A Framework for the Development of Domain-Specific Process Modelling Languages. In: Proceedings of the DESRIST 2017. , Karlsruhe, Germany (2017).
21. Keller, G., Nüttgens, M., Scheer, A.-W.: Semantische Prozeß modellierung auf der Grundlage “Ereignisgesteuerter Prozeß ketten (EPK),” (1992).
22. Cuntz, N., Kindler, E.: On the Semantics of EPCs: Efficient Calculation and Simulation. In: Nüttgens, M. and Rump, F.J. (eds.) Proceedings of the 3rd Workshop on Event-driven Process Chains (EPK2004). pp. 398–403. , Luxemburg (2004).
23. Valmari, A.: The state explosion problem. In: Lectures on Petri nets I: Basic models. pp. 429–528. Springer (1998).
24. OMG: Business Process Model and Notation (BPMN) Version 2.0. (2011).
25. BMJV: Wertpapierhandelsgesetz (WpHG). (1994).
26. Delfmann, P., Höhenberger, S.: Supporting Business Process Improvement through Business Process Weakness Pattern Collections. In: Proceedings of the 12. Internationale Tagung Wirtschaftsinformatik. pp. 378–392. , Osnabrück, Germany (2015).