vGMQL – Introducing a Visual Notation for the Generic Model Query Language GMQL

Matthias Steinhorst, Patrick Delfmann, and Jörg Becker

WWU Münster - ERCIS, Leonardo-Campus 3, 48149 Münster, Germany
{steinhorst, delfmann, becker}@ercis.uni-muenster.de

Abstract. This paper presents a visual query notation for the generic model query language GMQL. So far, GMQL allows for specifying pattern queries as complex set-theoretical formulas. This fact impedes the practical usability of GMQL, because specifying and understanding a query is unintuitive. The visual query notation we propose is a first step towards resolving this shortcoming. We derive objectives for this notation, implement it in a working prototype, as well as evaluate the notation using expert interviews and a literature survey.

Keywords: Conceptual Model Analysis, Enterprise Modeling, Pattern Matching, GMQL.

1 Introduction

The generic model query language GMQL has recently been proposed to query large collections of conceptual models [1]. Many companies have started to develop such collections as part of their business process management (BPM) [2] and enterprise modeling (EM) activities [3]. Examples of conceptual model collections include the SAP reference model with around 600 models [4], a model collection maintained by an Australian insurance company containing close to 7000 models [5], or the BIT process library containing about 700 models [6]. These examples demonstrate that such collections may indeed contain hundreds or even thousands of models [7].

Other than a form of documentation, conceptual models are a means of analyzing the aspect of corporate reality they capture in order to derive improvement potential. Given the size and complexity model collections may exhibit many practitioners have expressed the need for automatic or at least semi-automatic support of model analysis [8]. A task that frequently occurs in model analysis is querying a collection of models in order to detect certain patterns in them [7]. A pattern in this context refers to a model fragment that complies with a predefined pattern query.

Pattern detection serves a variety of analysis purposes ranging from model comparison [9-10] to model translation [11-12], model compliance checking [13], or model conflict detection [14]. GMQL was designed to support these model analysis tasks. GMQL is generic in the sense that it is able to query conceptual models of any type or graph-based modeling language. It is based on the idea that essentially any conceptual model is an attributed graph consisting of a set of nodes and a set of edges.

GMQL comes with a significant drawback: a pattern query is essentially a complex set-theoretical formula. Specifying as well as understanding a query is thus very cumbersome and unintuitive. The purpose of this paper is to introduce a visual query notation for GMQL to mitigate this shortcoming. We provide a visual shape for each GMQL construct and explain how these shapes can be used to visually model a pattern query. An initial survey of EM experts suggests that queries defined in the visual notation are much more intuitive to understand than the original formula-based pattern queries (see Section 5.1 for more details). The paper thus contributes to easing the usability of GMQL.

The remainder of this paper is structured as follows. First, we briefly introduce the core concepts behind GMQL. We then deduce objectives for a visual query notation from these concepts (Section 2). We introduce a visual shape for each GMQL construct in Section 3. We demonstrate the applicability of the visual version of GMQL that we call vGMQL by implementing it. We show the applicability of vGMQL by providing visual queries for model patterns presented in the BPM and EM literature (Section 4). We evaluate GMQL by first conducting a survey of EM experts to determine the understandability of the visual queries (Section 5.1). We then evaluate vGMQL against the backdrop of related work (Section 5.2). The paper closes with a summary and an outlook to future research in Section 6.

2 GMQL constructs and requirements for vGMQL

The basic idea of GMQL is that any graph-based conceptual model can be represented by two sets, namely the set O of its objects and the set R of its relationships (see [1] for an exact specification of all GMQL constructs). Objects denote the nodes of the model graph, whereas relationships represent its edges. We define the set of all model elements $E=O \cup R$. GMQL provides set-altering functions and operators that take these basic sets as input and perform certain operations on them. The GMQL functions fall into four classes. Functions belonging to the first class take one set of elements as input and return elements having particular characteristics like a specific type or label. The second class of functions determines elements having a particular number of (ingoing or outgoing) relationships (of a specific type). All functions return a set of sets with each inner set containing one element and all its relationships. Functions of the third class determine elements, their adjacent elements, as well as the relationship connecting these elements. The fourth class contains functions that determine paths or loops between two sets of elements. These paths may or must not contain particular model elements. Again, these functions return a set of sets with each inner set containing one path from one start element to one target element. As the theoretical roots of GMQL lie in set theory, it provides the basic set operators UNION, COMPLEMENT, and INTERSECT that perform denoted operations on two simple sets of elements. The JOIN operator performs a union on two sets if they have at least one element in common. INNERINTERSECT and INNERCOMPLEMENT perform respective operations on inner sets of a set of sets. As some operators and functions expect simple sets as input, the SELFUNION and SELFINTERSECT operators are necessary to turn one set of sets into a simple set while performing a

union or intersection. A GMQL pattern query is constructed by nesting these functions and operators. Pattern queries exhibit a tree-like structure with the output of one GMQL construct serving as input for the next. Based on this brief introduction of GMQL, requirements can be deduced for a visual query notation. Visual representations for all four classes of functions as well as all operators need to be defined. The notation should furthermore allow for nesting all GMQL constructs.

3 Conceptual specification

Figure 1 contains the visual representations for all four function classes (subsections A to F) as well as the operators (subsection G). The representation depicted in subsection A denotes an arbitrary element. It can be configured such that is represents an element having a particular type or label. If the shape is not further configured, it represents the set of all model elements. The shape is contained in all other vGMQL shapes represented in subsection B to F of Figure 1. It can furthermore be used as a placeholder for any other vGMQL shape including the operator shape. In doing so, it is possible to nest and concatenate the various constructs to construct pattern queries.

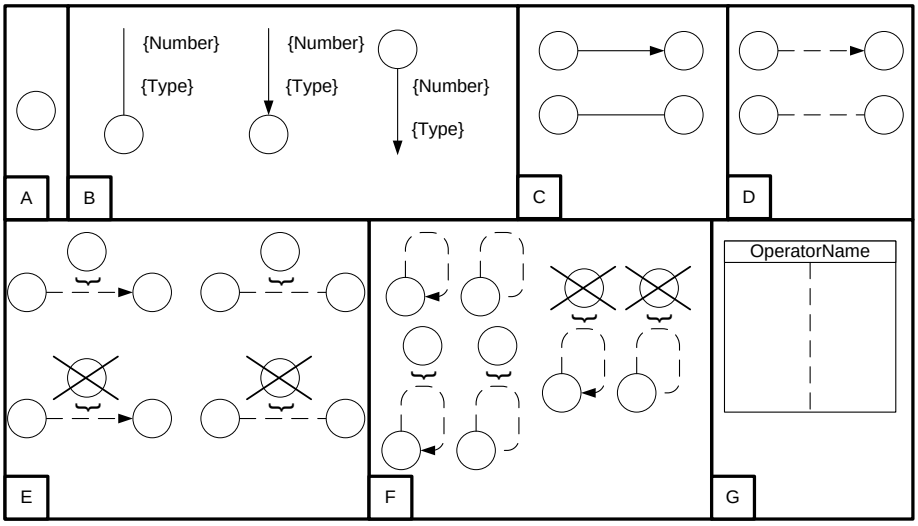

Fig. 1. vGMQL shapes

The shapes depicted in subsection B represent all functions of the second class returning single elements and all their relationships. The set R of all relationships is set to be the second input parameter for these functions. In case of directed edges, the relationships are represented by the outgoing and ingoing arrows. These functions return all relationships of a given element, even though their shapes include only one edge. The edges have two additional attributes called *Number* and *Type*. They indicate that the query is supposed to return elements having a particular number of relationships that are of a predefined type. If one of these attributes carries a NULL value, the shape represents the function taking only the other attribute as input.

The shapes depicted in subsection C of Figure 1 represent the functions of the third class returning adjacent elements and the relationships connecting them. Two different shapes for directed and undirected edges are provided. Note that these functions return all neighbors of a given element and the connecting relationships, although the shapes contain only one neighbor and relationship. Again, the shapes contain the basic element shape which allows for replacing it with any other combination of shapes (see more details below). In case the edges connecting the elements are represented as dotted lines, the corresponding shapes denote the paths functions (cf. subsection D of Figure 1). Different shapes are provided to represent functions for directed and undirected paths. The shapes depicted in subsection E represent functions for directed and undirected paths that must or must not contain specific elements. In case of the latter, the forbidden elements are crossed out. The shapes depicted in subsection F represent corresponding loop functions.

Lastly, subsection G of Figure 1 provides the visual shape for all operators. The name field can be customized to depict the corresponding operator names. The dotted line in the middle of the shape represents the two input parameters of each operator. In case the operator takes only one parameter as input, the line can be deleted.

4 Application examples and implementation

Figure 2 contains three exemplary vGMQL pattern queries for EPC diagrams (A and B) and ER models (C). The EPC queries are based on a language specification that only contains functions, events, as well as AND, OR, and XOR connectors as object types. The ERM pattern is based on a language specification containing only entity types and relationship types. All language specifications furthermore contain the respective relationship types. The pattern query in subsection A of Figure 2 represents a conflict pattern in EPCs reported by Mendling [14] who refers to this structure as an "AND join that might not get control from a splitting XOR". It represents a situation in which an AND following an EPC start event is the target element of a path that starts in an XOR split. If the start event fires and the process has run into a branch other than the one containing the AND connector, this AND will never be executed.

The pattern query depicted in subsection A contains the directed path function as its outermost shape. The first input parameter of the function represents a splitting XOR connector. It is calculated by subtracting the set of all XOR nodes having one outgoing edge from the set of all XORs. The set of XOR nodes having one outgoing edge is inner-intersected with all XORs to cut off the edge. The second parameter represents an AND join that is following an EPC start event. To that end, the shape representing adjacent elements is used. The first input parameter represents the set of all EPC start events. It is calculated by inner-intersecting the set of all events with the set of events having zero ingoing edges. To turn the resulting set of sets into a simple set the SELFUNION operator is used. The second input parameter represents a joining AND connector that is calculated analogously to a split node. This sub-query thus returns an event with no ingoing edges that is followed by a joining AND connector. This structure is inner-intersected with the set of all ANDs in order to cut

off the start event as well as the relationships connecting the event and the connector. The remaining AND object is fed to the path function as its second input parameter.

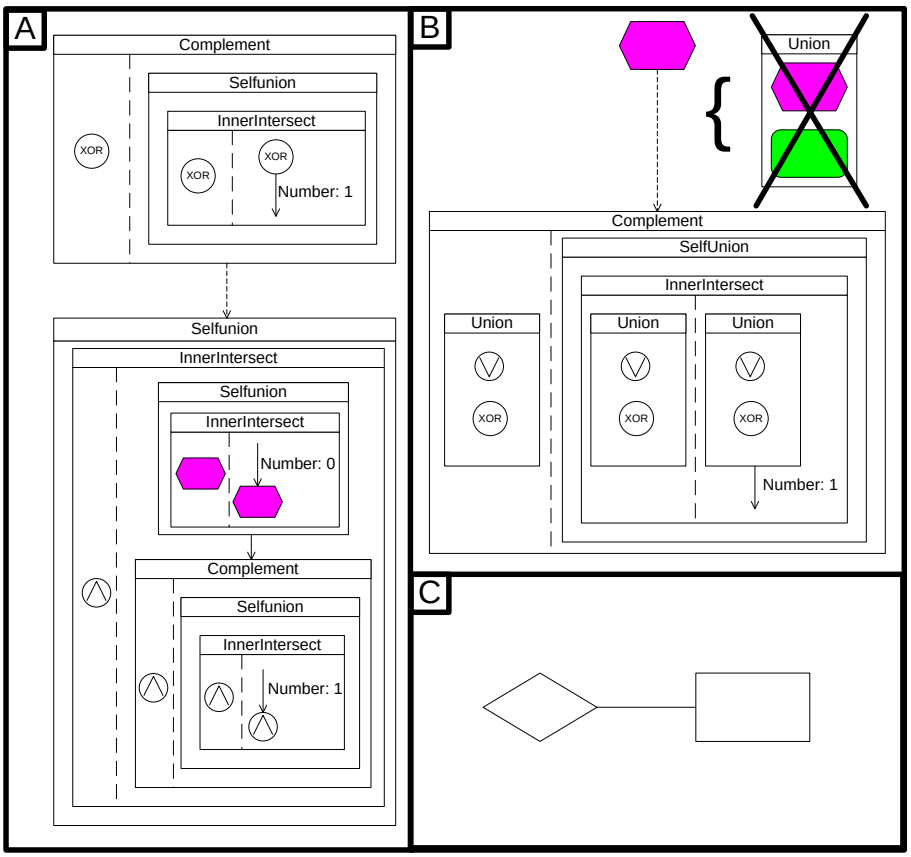

Fig. 2. vGMQL pattern queries for EPCs and ER models

The pattern query depicted in subsection B of Figure 2 represents a common syntactical error in EPC models. This error consists of a decision split after an event. This pattern can be described as an element path that starts in an event object and ends in either an OR or XOR split such that the path only contains connector objects. Functions and events are thus not allowed on this path. To define such a pattern query in vGMQL, the shape representing a path that must not contain particular elements is used. The first parameter represents the set of all event objects. The second parameter represents the set of all decision splits. Again, this sub-query is calculated analogously to the corresponding split-query described above. The only difference is that we are interested in the unified set of all XOR and OR connectors. The third parameter represents the set of all forbidden elements.

The pattern query in subsection C of Figure 2 represents an ERM relationship type that is adjacent to one or more entity types. This query thus returns binary and ternary relationship types.

Figure 3 depicts a prototypical implementation of the visual notation in a query editor. The original GMQL was implemented as a plugin for a meta-modeling tool that was available from a previous research project. The meta-modeling tool allows for specifying modeling languages by defining its object and relationship types. Similar to vGMQL the tool is based on the idea that any modeling language can be represented as the set of its element types. To develop a model, the element types of the corresponding language are instantiated to a set of elements that is used to calculate the basic sets O and R that vGMQL requires for its matching procedure. On meta-level the meta-modeling tool is thus based on the same concept that vGMQL uses to detect patterns in models. This fact allows vGMQL to be language-independent, because pattern queries can be defined for all modeling languages that can be specified using the meta-modeling tool.

The pattern matching functionality provided by vGMQL is integrated into the language editor of the tool which contains functionality for specifying languages. For each defined modeling language pattern queries can be created. All vGMQL shapes are provided on the left-hand side of the editor. The user can drag and drop these shapes on the query editing field on the right-hand side of the editor. The pattern query depicted in Figure 3 represents the EPC syntax error "decision split after event" as described above. As demonstrated in the figure, all vGMQL shapes can take other shapes as input. This allows for nesting the constructs of the query language in order to construct complex query definitions. Upon saving a pattern query, it is parsed into the original formula-based representation that is then fed to the matching mechanism. This mechanism is implemented using the visitor design pattern known from software engineering [15]. A visitor object thus traverses the query-tree in a bottom-up fashion calculating the leaf nodes of the tree first. The corresponding result serves as input for the next higher tree level.

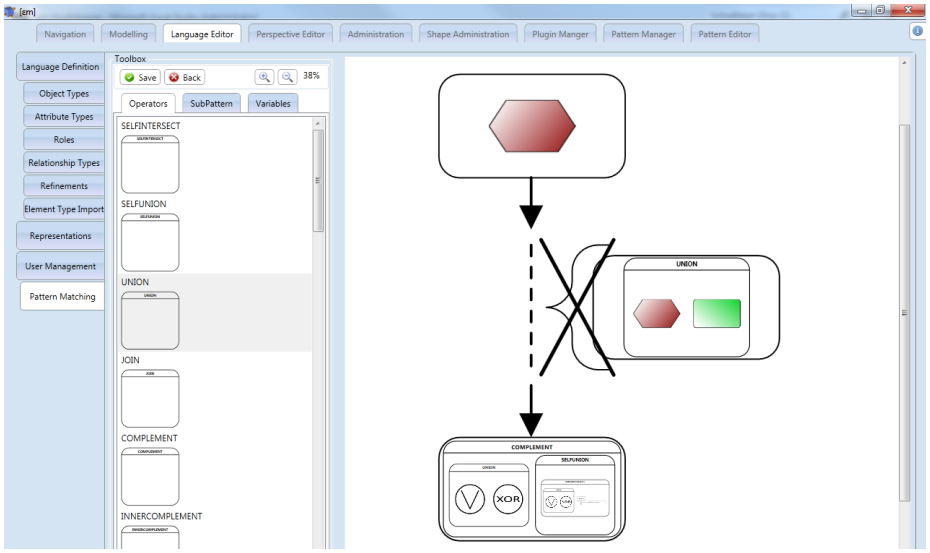

Fig. 3. vGMQL implementation

5 Evaluation

5.1 Survey

To evaluate the visual query notation, we conducted an initial survey of ten EM experts having between one and seven years of work experience. To guarantee an unbiased feedback, the experts did not have any prior knowledge of GMQL and were thus briefly introduced to its underlying concepts. The participants of the survey were then given the EPC syntax error representing a decision split after an event. We presented both the formula-based query as well as its visual counterpart (cf. Figure 2, subsection B) to the participants. They were then asked which of the queries they perceived to be more intuitive to understand. The set of possible answers also included the possibility to express that both queries are equally understandable.

Out of the ten experts involved in this initial survey, seven voted for the visual query and one participant found the formula-based query to be more intuitive to understand. Two participants furthermore perceived both queries to be equally understandable. The participant who found the formula-based query to be more intuitive argued that he is used to reading source-code and thus found the original GMQL query to be easier to understand. One participant who voted in favor of the visual query furthermore argued that the original formula-based query would potentially be as intuitive to understand as its visual counterpart provided an EM analyst possesses the necessary in-depth knowledge of the set-theoretical functions and operators. Given the results of this initial survey, we preliminarily conclude that the visual query notation we propose in this paper indeed eases the usability of GMQL, because visual queries appear to be more intuitive to understand than the original formula-based queries. Future surveys including larger sets of participants as well as pattern queries need to confirm this finding. In addition, this initial survey is limited to comparing the understandability of two given pattern queries. Additional surveys also need to focus on the perceived ease of defining queries in order to provide a complete picture of the language's usability.

5.2 Related work

vGMQL is primarily designed for a structural model analysis. vGMQL, however, is able to consider element types and labels in its matching process. Analyzing element labels is difficult, because studies indicate that conceptual models can vary significantly with respect to terms and phrase structures used to label model elements [16]. This impedes the applicability of conceptual models, because different user groups may understand particular terms differently. This in turn also impedes the applicability of vGMQL, because searching for a particular pattern containing a given label will not return all results if labels contain semantic ambiguities. Prior to searching for patterns with vGMQL it is therefore necessary to terminologically standardize labels in order to avoid semantic ambiguities like synonyms, homonyms, etc. Corresponding approaches [17-18] need to be integrated into vGMQL.

vGMQL is furthermore not designed for analyzing the execution semantics of process models. This can be achieved using finite transition systems [19] or behavioral profiles [20]. We refer to respective literature on analyzing process model execution semantics. As vGMQL includes element types in its matching process, the path functions, however, can be used to detect simple patterns representing violations to specific runtime constraints (see [1] for more detailed examples). Extending vGMQL to include process model execution semantics, however, remains subject of future research.

vGMQL furthermore assumes that there is a predefined pattern query available that can be searched for in a given collection of models. It is thus not suited for analysis scenarios in which this is not the case. Consider for example the work put forth by [21] to identify exact clones in a collection of process models. A clone represents a particular model fragment (i.e., pattern) that frequently occurs in the collection. The algorithm proposed by [21] is able to iteratively construct these patterns without having a predefined query fragment to search for. vGMQL is consequently suited for analysis scenarios in which predefined pattern queries are available. Notable examples presented in the literature include model comparison [9], model compliance checking [13], model weakness detection [22], model abstraction [23], model syntax checking [24], or model refactoring [25].

From a graph-theoretical point of view, the problem of pattern matching can be understood as the problem of subgraph isomorphism. As this problem is known to be NP-complete in the general case [26], the runtime performance of respective algorithms is a primary concern. [27] extend the well-known Ullmann algorithm for subgraph isomorphism [28] to include a filter mechanism that reduces the number of models to be searched for a given pattern. Subgraph isomorphism, however, is concerned with finding exact occurrences of a given pattern in a model. In the context of pattern matching in conceptual models, it is often necessary to find paths of previously unknown length. vGMQL provides this functionality and can thus be more broadly applied than algorithms for subgraph isomorphism.

Lastly, additional multi-purpose process query languages haven been proposed in the literature. Notable examples include BPMN-Q [29], BPQL [30], and BP-QL [31]. vGMQL differs from these approaches as it can not only search process models but also models of any other type or graph-based modeling language. With this paper, we furthermore present a visual notation that allows for visually specifying a query similarly to respective approaches presented in the literature.

6 Summary and outlook

In this paper, we presented a visual query notation for the multi-purpose and language-independent model query language GMQL. Specifying pattern queries thus no longer requires constructing complex set-theoretical formulas. Future research will focus on conducting additional surveys and experiments with EM experts to further test whether this notation is indeed easier to use than the original formula-based one. In addition, we will conduct a survey among modeling experts to determine the applicability of vGMQL in the context of specifying large and complex queries. We

will also compare the proposed visualization to alternative ways of graphically modeling pattern queries. This will carve out additional user needs and determine the most intuitive way of formulating a pattern query. We will furthermore explore additional enterprise modeling related usage scenarios of the query language like ad hoc error and inconsistency detection during model development.

References

1. Bräuer, S., Delfmann, P., Dietrich, H., Steinhorst, M. 2013. "Using a Generic Model Query Approach to Allow for Process Model Compliance Checking — An Algorithmic Perspective," in Proceedings of the 11th International Conference on Wirtschaftsinformatik, Leipzig, Germany, pp. 1245–1259.
2. Rosemann, M. 2006. "Potential pitfalls of process modeling: Part a," in Business Process Management Journal (12:2), pp. 249–254.
3. Stirna, J., Persson, A. 2012. "Evolution of an Enterprise Modeling Method – Next Generation Improvements of EKD," in Proceedings of the 5th IFIP WG8.1 Working Conference on the Practice of Enterprise Modeling, Rostock, Germany, pp. 1-15.
4. Keller G., Teufel T. 1998. SAP R/3 process-oriented implementation: Iterative process prototyping, Harlow: Addison Wesley Longman.
5. La Rosa, M., Reijers, H.A., van der Aalst, W.M.P., Dijkman, R., Mendling, J., Dumas, M., and García-Bañuelos, L. 2011. "APROMORE: An Advanced Process Model Repository," in Expert Systems with Applications (38:6), pp. 7029-7040.
6. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K. 2009. "Instantaneous soundness checking of industrial business process models," in Proceedings of the 7th International Conference on Business Process Management, Ulm, Germany, pp. 278-293.
7. Dijkman R., La Rosa, M., and Reijers, H.A. 2012. Managing Large Collections of Business Process Models: Current Techniques and Challenges," in Computers in Industry (63:2), pp. 91-97.
8. Houy, C., Fettke, P., Loos, P., van der Aalst, W.M.P., and Krogstie, J. 2011. "Business process management in the large," Business and Information Systems Engineering (3:6), pp. 385–388.
9. Yan, Z., Dijkman, R., and Grefen, P. 2010. "Fast Business Process Similarity Search," Distributed and Parallel Databases (30:2), pp. 105-144.
10. La Rosa, M., Dumas, M., Uba, R., and Dijkman, R. 2013. "Business Process Model Merging: An Approach to Business Process Consolidation," Transactions on Software Engineering and Methodology (22:2), in print.
11. Ouyang, C., Dumas, M., ter Hofstede, A.H.M., and van der Aalst, W.M.P. 2007. "Pattern-based Translation of BPMN Process Models to BPEL Web Services," International Journal of Web Services Re-search (5:1), pp. 1-21.
12. García-Bañuelos, L. 2008. "Pattern Identification and Classification in the Translation from BPMN to BPEL," in Proceedings of the Confederated International Conferences on the Move to Meaningful Information Systems, R. Meersmann and Z. Tari (eds.), Monterrey, Mexico, pp. 436-444.
13. Awad, A., Decker, G., and Weske, M. 2008. "Efficient Compliance Checking Using BPMN-Q and Temporal Logic," in Proceedings of the 6th International Conference on Business Process Management, M. Dumas, M. Reichert, and M.-C. Shan (eds.), Milan, Italy, pp. 326-341.

14. Mendling, J. 2007. Detection and Prediction of Errors in EPC Business Process Models, Doctoral Thesis, Vienna University of Economics and Business Administration. Vienna, Austria.
15. Gamma, E., Helm, R., Johnson, R. E.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Longman, Amsterdam (1995).
16. Hadar, I., and Soffer, P. 2006. "Variations in Conceptual Modeling: Classification and Ontological Analysis," Journal of the Association for Information Systems (7:8), pp. 568-592.
17. Thomas, O., and Fellmann, M. 2009. "Semantic Process Modeling – Design and Implementation of an Ontology-based Representation of Business Processes," Business and Information Systems Engineering (1:6), pp. 438-451.
18. 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, J.F. Nunamaker Jr., W.L. Currie (eds.), Phoenix, USA, Paper 45.
19. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A. 1998. Deriving Petri Nets from Finite Transition Systems, IEEE Transactions on Computers (47:8), pp. 859-882.
20. Weidlich, M., Mendling, J., Weske, M. 2011. "Efficient Consistency Measurement Based on Behavioral Profiles of Process Models," IEEE Transactions on Software Engineering (37.3), pp. 410-429.
21. Dumas, M., García-Bañuelos, L., La Rosa, M., and Uba, R. 2013. "Fast detection of exact clones in business process model repositories," Information Systems (38:4), pp. 619-633.
22. Becker, J., Bergener, P., Räckers, M., Weiß, B., and Winkelmann, A. 2010. "Pattern-Based Semi-Automatic Analysis of Weaknesses in Semantic Business Process Models in the Banking Sector," in Proceedings of the 18th European Conference on Information Systems, T. Alexander, M. Turpin, and JP van Deventer (eds.), Pretoria, South Africa, Paper 156.
23. Polyvyanyy, A., Smirnov, S., and Weske, M. 2010. "Business Process Model Abstraction," in Handbook on Business Process Management 1: Introduction, Methods and Information Systems, M. Rosemann and J. vom Brocke (eds.), New York: Springer-Verlag, pp. 149-166.
24. Gruhn, V., Laue, R., Kühne, S., Kern, H. 2009. "A Business Process Modelling Tool woth Continuous Validation Support," Enterprise Modelling and Information Systems Architectures (4:2), pp. 37-51.
25. Weber, B., Reichert, M., Mendling, J., Reijers, H.A. 2011. "Refactoring large process model repositories," Computers in Industry (62:5), pp. 467-486.
26. Garey, M.R., and Johnson, D.S. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness, New York: W. H. Freeman & Co.
27. Jin, T., Wang, J., Wu, N., La Rosa, M., and ter Hofstede, A.H.M. 2010. "Efficient and Accurate Retrieval of Business Process Models through Indexing", in Proceedings of the Confederated International Conferences on the Move to Meaningful Information Systems, R. Meersman, T. Dillon, P. Herrero, P. (eds.), Crete, Greece, pp. 402-409.
28. Ullmann, J.R. 1976. "An Algorithm for Subgraph Isomorphism," Journal of the Association of Computing Machinery (23:1), pp. 31-42.
29. Awad, A. 2007. "BPMN-Q: A Language to Query Business Processes," in Proceedings of the 2007 Work-shop on Enterprise Modelling and Information Systems Architectures, M. Reichert, S. Strecker, K. Turowski (eds.), St. Goar, Germany, pp. 115-128.
30. Momotko, M., and Subieta, K. 2004. "Process Query Language: A Way to Make Workflow Processes More Flexible," in Proceedings of the 8th East European Conference on Advances in Databases and In-formation Systems, A. Benczúr, J. Demetrovics, G. Gottlob, (eds.), Budapest, Hungary, pp. 306-321.
31. Beeri, C., Eyal, A., Kamenkovich, S., and Milo, T. 2008. "Querying business processes with BP-QL", Information Systems (33:6), pp. 477-507.