A Web-Based Tool for Diagramed Model Query

Sabine Nagel*, Carl Corea and Patrick Delfmann

University of Koblenz, Universitätsstr. 1, 56070, Koblenz, Germany

Abstract

We present a web-based tool for querying conceptual process models using the Diagramed Model Query Language (DMQL 2.0). Users can graphically design query patterns, upload process models in BPMN, EPC, or organizational chart formats, and interactively explore matched results. The tool supports advanced features including path constraints, sub-patterns, and cross-notation queries. We evaluated our tool across multiple iterations, with the last iteration being the application of the tool in a Master's course to interactively teach model query.

Keywords

DMQL, Model Query, Process Modeling, Pattern Matching

1. Introduction

Organizations commonly maintain large repositories of conceptual models, such as process models, data models, and organizational charts, to document, analyze, and improve their information systems [1]. As these repositories grow, manual inspection becomes impractical, and automated querying is essential for tasks such as business process compliance management and structural weakness detection [2, 3]. To this end, model query languages allow analysts to specify a structural pattern and automatically retrieve every model fragment that matches it [4].

The Diagramed Model Query Language (DMQL) was developed to address key limitations of existing query approaches [1]. Most languages are restricted to a single modeling notation, rely on text-based or formula-based query specification that domain experts find difficult to use, and require model transformations before queries can be executed (cf. [4]). DMQL overcomes these limitations by operating on a modeling-language-agnostic attributed multigraph representation, and by allowing analysts to specify queries by drawing a diagram that visually resembles the model fragment being searched for. DMQL has since been extended to DMQL 2.0 [2], which adds support for negation and universal operators through forbidden vertices, forbidden edges, and forbidden paths.

The original implementation of DMQL was provided as a plug-in for the desktop meta-modeling tool [εm] [2]. While this demonstrated the feasibility of DMQL in real-world scenarios, its desktop-only architecture imposes practical barriers and limitation to specific operating systems. Furthermore, the underlying technologies and implementation techniques have become outdated and no longer reflect the capabilities of modern web-based modeling environments. To address these limitations, we present a web-based tool for diagramed model query that requires no installation and is accessible directly in the browser. The remainder of this paper is structured as follows: Section 2 introduces the formal foundations of DMQL. Section 3 describes the tool and its functionality, while Section 4 reports on usage and maturity. We conclude with Section 5.

2. Preliminaries

The Diagramed Model Query Language (DMQL) is a structural query language to search conceptual models for substructures that match a specified pattern [2, 1]. Its foundation is a modeling-language-

Proceedings of the BPM 2026 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum (BPM 2026), Toronto, Canada, September 27th to October 2nd, 2026.

*Corresponding author.

snagel@uni-koblenz.de (S. Nagel); ccorea@uni-koblenz.de (C. Corea); delfmann@uni-koblenz.de (P. Delfmann)

0000-0003-4838-8246 (S. Nagel); 0000-0001-6420-1272 (C. Corea); 0000-0003-4441-0311 (P. Delfmann)

agnostic meta-model in which any conceptual model is treated as an attributed multigraph.

Definition 1 (Conceptual Model)**.** A conceptual model is a tuple $M = (V, E, T_V, T_E, C, \alpha, \beta, \gamma)$, where V is a set of vertices (modeling objects such as activities or events), $E = E_D \cup E_U$ is a set of edges with $E_D \subseteq V \times V \times \mathbb{N}$ being directed edges and $E_U = \{\{v_1, v_2, n\} \mid v_1, v_2 \in V, n \in \mathbb{N}\}$ being undirected edges, T_V and T_E are sets of vertex and edge types, C is a set of captions (labels and attribute values), and $\alpha : V \mapsto T_V$, $\beta : E \mapsto T_E$, $\gamma : V \cup E \mapsto C$ are functions that assign each element its type and caption.

DMQL queries this graph representation by means of a *query pattern*, which is itself a graph whose vertices and edges carry properties that specify what the matched substructure must look like.

Definition 2 (DMQL Query Pattern)**.** A DMQL query pattern is a tuple $Q = (V_Q, E_Q, P_V, P_E, \delta, \varepsilon, \Gamma, G)$, where V_Q is a set of query vertices, $E_Q \subseteq V_Q \times V_Q \times \mathbb{N}$ is a set of query edges, and $\delta : V_Q \to P_V$ and $\varepsilon : E_Q \to P_E$ are functions that assign properties to each query vertex and edge, respectively.

Vertex properties are tuples $P_V \subseteq VID \times VCAPTION \times VTYPES$, where *vid* is a unique identifier, $vcaption \in C$ is a caption matching phrase, and $vtypes \subseteq T_V$ is a set of admissible vertex types.

Edge properties are tuples $P_E \subseteq EID \times ECAPTION \times DIR \times MINL \times MAXL \times MINVO \times MAXVO \times MINEO \times MAXEO \times VTYPESR \times VTYPESF \times ETYPESR \times ETYPESF \times \Theta$. *eid* and *ecaption* are analogous to their vertex counterparts and $dir \subseteq \{org, opp, none\}$ specifies whether an edge must follow the original, opposite, or no direction. *minl* and *maxl* define the minimum and maximum path length to which the query edge may be mapped. *minvo*, *maxvo*, *mineo*, and *maxeo* control how often a matched path may revisit the same vertices or edges and *vtypesr*, *vtypesf*, *etypesr* and *etypesf* specify vertex and edge types that are required or forbidden on a matched path. Lastly, Θ allows the embedding of sub-patterns that must or must not occur within a matched path and G is a set of global rules that can impose additional constraints on vertex and edge properties, such as matching captions.

The matching algorithm searches a target model for all subgraphs in which query vertices map to model vertices and query edges map to model edges or paths that satisfy the specified constraints [1]. Because this formalization operates on the modeling-language-agnostic meta-model of Definition 1, DMQL can be applied uniformly to process models across different notations without modification. The syntax of DMQL is designed so that a query visually resembles the model fragment being searched for (cf. Figure 1), which makes query specification substantially more intuitive than text-based alternatives. This design supports a range of analysis tasks, including compliance checking against regulatory rules, detection of structural weaknesses, and search across large process model repositories.

3. Tool Description

In this work, we introduce a web-based tool for designing and executing DMQL queries on process models. It is built on the formal foundations of DMQL 2.0 [2] and supports querying across BPMN, EPC, and organizational chart notations through a four-step workflow: (1) language selection, (2) graphical query design, (3) model upload, and (4) interactive results exploration. The front-end is implemented in Angular 19 with JointJS for interactive canvas rendering; the backend is a Python FastAPI service that executes pattern matching via the subgraph isomorphism algorithm from [1].

3.1. Query Pattern Designer

The query design phase begins with the selection of languages, i.e., users can choose between BPMN, EPC, organizational charts, or any combination to enable cross-language pattern matching. This selection determines the vocabulary available for pattern construction, specifically, the set of element types, edge types, and properties that can be used in the query pattern. Once languages are selected, users proceed to the visual query designer, which provides an interactive canvas where query patterns can be constructed by adding nodes (representing process elements) and edges (representing relationships) (cf. Figure 1). The designer supports a comprehensive set of pattern definition capabilities.

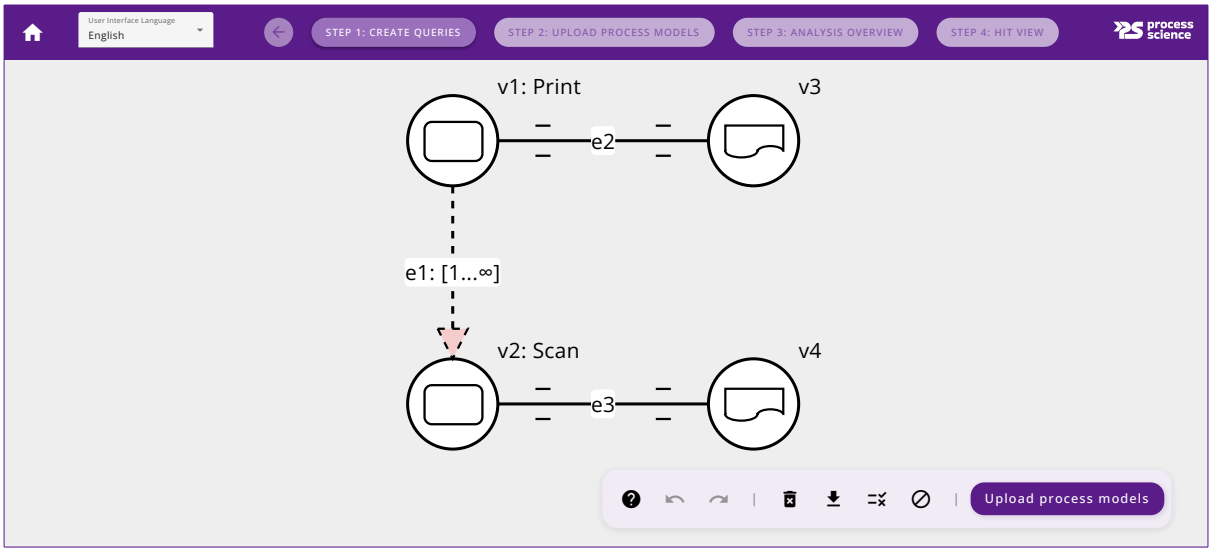

Figure 1: Query Pattern Designer

For nodes, users can specify which element types from the selected language are acceptable matches. The tool displays language-specific element palettes with icons and respective descriptions. Each node can be marked as optional (weak constraint), required, or forbidden. Weak constraints allow elements to be absent from valid matches, while forbidden elements function as anti-patterns that reject any match containing them. Furthermore, users can attach caption patterns based on regular expressions.

Edges offer more advanced configuration options. Users can specify directional constraints (forward, backward, or undirected) and enable path mode to represent sequences of multiple intermediate elements. In path mode, users can define minimum and maximum path lengths to constrain which sequences are acceptable. Overlap settings control whether vertices and edges can be visited or traversed multiple times during matching. Sub-patterns can be embedded within path edges to provide hierarchical pattern composition, with the possibility to mark subpatterns as required (must occur), partially required (must occur in at least one match), allowed (optional occurrence), partially forbidden (must not occur in all matches), or forbidden (must not occur at all). This hierarchical capability enables reuse of patterns.

Global rules extend pattern matching beyond structural constraints to semantic properties. Users can define property-based rules with support for combining multiple rules using logical operators (AND, OR, XOR). Additionally, users can configure settings that control how the tool treats patterns that contain forbidden elements. Users can require all forbidden elements to be present, at least partially present, or configure the tool to reject matches if any single forbidden element is fully or partially present.

Queries can also be exported, which allows for iterative refinement and reuse in future analyses.

3.2. Analysis and Results Exploration

After designing a query pattern, users can upload one or more process models for analysis. The tool accepts BPMN files (exported in XML format from BPMN.io), EPC files, and organizational charts (both in SAP Signavio export format). Format-specific translators parse each file type and construct the tool's internal attributed multigraph representation, where vertices represent model elements, directed and undirected edges represent relationships, vertex types and edge types distinguish element classes, and captions store labels and attribute values. This unified representation enables the pattern matching algorithm to operate identically across all supported notations.

Next, the matching algorithm efficiently discovers all subgraphs in the process models that match the query pattern. The results are presented through an interactive visualization interface (cf. Figure 2). A sidebar displays a preview of the query pattern to show what was searched for, along with a grid of numbered matches that represent each discovered pattern occurrence. Users can click on

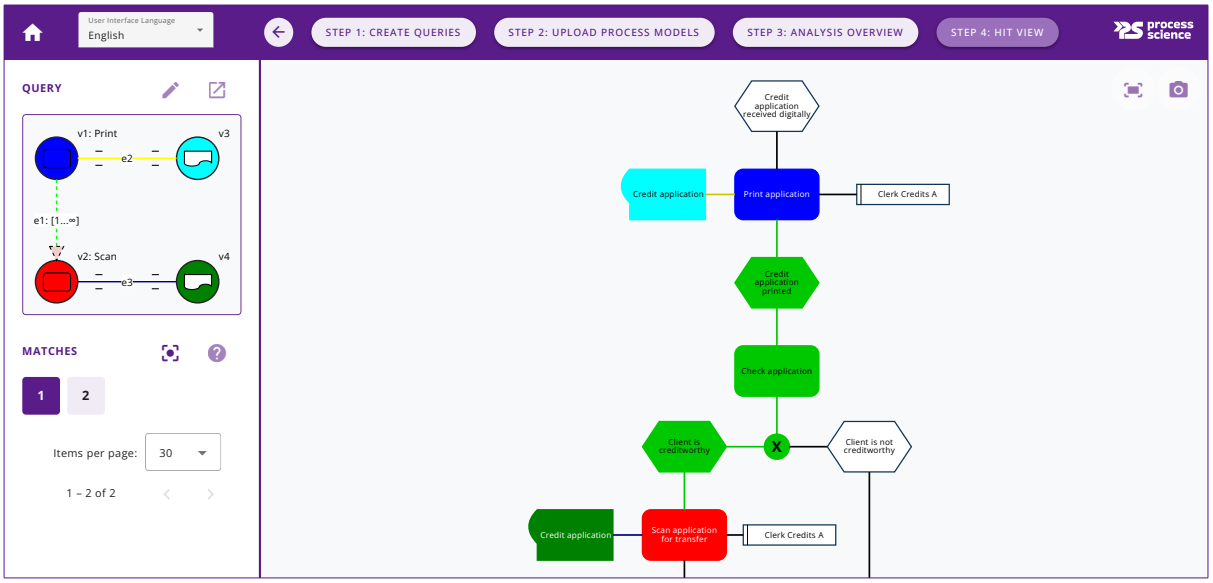

Figure 2: Results Exploration in Hit View

matches to navigate sequentially through the results, and the main canvas automatically highlights the corresponding matched elements in the process model. For cross-language queries, the interface presents a split-screen view showing two process models side-by-side.

Additionally, users can export an image of the currently visible result (i.e., the process model with the currently selected match highlighted).

4. Usage and Maturity

The tool does not require installation and is accessible directly in the browser[1]. A screencast demonstrating the tool's functionality is available online[2].

We evaluated the tool across three successive development iterations, with each iteration targeting distinct aspects of the tool's functionality and design. The first iteration concentrated on establishing the core functionality of DMQL 1.0, while the second iteration extended these functionalities, incorporated revisions based on initial feedback, and introduced additional features related to DMQL 2.0. The third iteration shifted emphasis toward usability, including major layout revisions, usability fixes based on the previous iteration's evaluation, and a substantially redesigned and interactive hit view. The evaluation of the third iteration was conducted in a master's-level BPM course, with six students participating. Participants were initially asked to complete a DMQL-related assignment by formalizing and drawing patterns manually on paper. Next, participants were granted access to the tool and instructed to revisit the same assignment, with the opportunity to revise their solutions based on their experience with the tool. To systematically capture participant perceptions throughout this process, questionnaires were provided both prior to and following tool use to address dimensions such as conceptual understanding of DMQL, confidence in submitted answers, approaches to pattern verification, and overall impressions of the tool. In addition, usability was assessed using the System Usability Scale (SUS), a well-established standardized instrument for evaluating perceived usability across interactive systems [5].

The tool received a SUS score of 74.17, which suggests that participants perceived the tool as broadly usable, which is also supported by the collected qualitative responses. Several participants reported an increase in confidence after using the tool, with one noting that feelings of uncertainty "vanished after using the tool" and another positively highlighting the ability to "test my hypotheses very quickly". The functionality to upload an EPC and observe which elements of a pattern were matched was frequently

[1]https://uni-ko.de/dmql-tool
[2]https://uni-ko.de/dmql-screencast

cited as valuable, as it provided a direct and interactive means of refining pattern understanding. Improvements in conceptual understanding were also mentioned, with one participant stating that "the notation is easier to understand after using the tool." Although some participants encountered initial difficulties, most reported that these vanished once they had spent some time exploring the tool.

5. Conclusion

In this work, we introduced a web-based tool for diagramed model query based on DMQL 2.0 that enables users to define and execute model queries across different conceptual modeling languages, including BPMN, EPC, and organizational charts. By offering browser support, the tool makes advanced model query accessible to users without the need for complex setup or system restrictions. Furthermore, our tool allows users to define queries in a visual way (rather than queries based on formal expressions), which further lowers the barrier for applied model query. Our evaluation demonstrated that users perceived the tool as usable and beneficial for creating and understanding query patterns and results.

In future work, we aim to extend the tool with further collaborative features and modeling notations, as well as to continuously evaluate the tool in both education and real-world model query scenarios.

Acknowledgments

The first two iterations of the tool were developed in the scope of research labs at the University of Koblenz, so we thank all involved students for their contribution.

Declaration on Generative AI

The author(s) used Sonnet 4.6 for grammar and spelling check, as well as Microsoft Copilot and Claude Code for error handling and code cleanup during several implementation iterations.

References

[1] P. Delfmann, D. Breuker, M. Matzner, J. Becker, Supporting Information Systems Analysis Through Conceptual Model Query – The Diagramed Model Query Language (DMQL), Communications of the Association for Information Systems 37 (2015).

[2] P. Delfmann, D. M. Riehle, S. Höhenberger, C. Corea, C. Drodt, The Diagramed Model Query Language 2.0: Design, Implementation, and Evaluation, in: A. Polyvyanyy (Ed.), Process Querying Methods, Springer International Publishing, Cham, 2022, pp. 115–148.

[3] T. Käuter, P. Stünkel, A. Rutle, Y. Lamo, H. König, Bpmn analyzer 2.0: Instantaneous, comprehensible, and fixable control flow analysis for realistic bpmn models, in: Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 22nd Int. Conference on Business Process Management (BPM 2024), volume 3758 of *CEUR Workshop Proceedings*, Krakow, Poland, 2024, pp. paper–11.

[4] A. Polyvyanyy, Introduction to process querying, in: A. Polyvyanyy (Ed.), Process Querying Methods, Springer International Publishing, Cham, 2022, pp. 1–18.

[5] J. Brooke, et al., Sus - a quick and dirty usability scale, Usability evaluation in industry 189 (1996) 4–7.