Outlining a Graphical Model Query Approach Based on Graph Matching

Dominic Breuker, Hanns-Alexander Dietrich, Matthias Steinhorst, Patrick Delfmann

European Research Center for Information Systems (ERCIS)
University of Münster
Leonardo-Campus 3, 48149 Münster, Germany
{breuker|dietrich|steinhorst|delfmann}@ercis.uni-muenster.de

Abstract: This paper outlines a graphical model query approach based on graph matching. It consists of a graphical query specification language and a matching algorithm based on graph matching that takes the query as input and returns all matches found in a model to be searched. The graphical query specification language can be used to draw model queries much like a model would be constructed. To achieve applicability in many different model analysis scenarios, the query approach provides structurally exact and structurally similar pattern matching as well as semantic comparison of model node and edge contents. Following a design science research process, we derive functional requirements for the query language and matching algorithm from the literature, outline its syntax, formally specify its matching principle, and demonstrate its functionality by providing a working prototype implementing previously identified requirements.

1 Introduction

Conceptual models are labeled and typed graphs that can be used to describe and analyze an organization and its information systems. Examples of conceptual models are process models that describe the order in which a set of business activities are executed, data models that capture the data necessary to execute business tasks, or organizational charts representing the relationships of employees and departments of a company. Conceptual models do not only serve to document but also to analyze specific aspects of corporate reality. In this context, we understand "analysis" as the task of extracting relevant structural and semantic (i.e., label) information out of conceptual models for a given task. In many cases, such an analysis results in querying a collection of models to identify model fragments with (partly) given structural characteristics and (partly) given contents. Such an analysis may serve various business objectives (for a detailed discussion on model analysis purposes, including different kinds of conceptual models, cf. [De14]):

- One reason for analyzing conceptual models is *compliance checking* against laws and regulations [Aw07]. Compliance management has become an important management task that – if done incorrectly or not at all – can be very costly for an organization. Compliance checking requires identifying model fragments, either in process models only or in combination with other types of models (e.g., data models, organizational charts, etc.), whose structure and contents indicate a compliance violation [Kn10].

- *Identifying weaknesses in process models* serves to improve business processes according to efficiency, effectiveness, or quality. It aims at avoiding double work or unnecessary manual processing, for instance [Bec10]. Identifying such weaknesses requires finding process model fragments whose structure and contents typically indicate a shortcoming of a business process.
- Model translation is about *transforming conceptual models* from one notation into another one, for instance from a conceptual into an executable specification. Notable examples discussed in the literature include translating BPMN models into BPEL code [ODA08, Ga08] or transforming data models into relational schemas. Model translation requires identifying model fragments of a source notation that are to be translated to model or code fragments of a target notation.
- A further purpose of model analysis is *checking models for structural or behavioral conflicts* to ensure their syntactical correctness and – in case of process models – their proper execution semantics [Me08]. Syntax errors or improper execution semantics may disrupt runtime execution. Hence, identifying such problems can help assuring proper model execution in the case of automation. Identifying structural or behavioral conflicts requires identifying model fragments that indicate such conflicts.

Querying conceptual models to identify fragments with particular characteristics thus serves an abundance of analysis tasks that are performed to design, redesign, or improve corporate information systems in different ways. Hence, many companies have started to create large collections of conceptual models [DRR12]. Such collections mostly include process models but may also include other types of conceptual models like data models, organizational charts, or ontologies. They may contain hundreds to thousands of models and, in turn, each model may consist of hundreds to thousands of elements [Di11]. Given the complexity of these collections, the task of analyzing conceptual models is becoming increasingly difficult [Di11]. Analyzing conceptual models is even more complex considering that such an analysis may serve different business objectives as described above. The scientific community has put forth a great number of different model query approaches (cf. Section 6). They allow for querying a model collection automatically in order to identify particular pattern matches in the models. A pattern match in this context refers to a model fragment that complies with a predefined query pattern defining the fragment's structure and labels.

A common characteristic of many recent model query approaches is that they are specifically designed to support one particular business objective (e.g., [YDG12]). In addition, some of these approaches are designed for analyzing models developed in a particular modeling language (e.g., [Aw07] or [Be08]). Instead of developing such customized query approaches, we follow the argument of [Aa13] arguing that it is more beneficial to develop a query approach that can be used to support multiple business objectives. Furthermore, we argue that it is beneficial to develop an approach that is by default applicable to multiple types of conceptual models (i.e., process models, data models, etc.) and models of any modeling language. Companies that wish to automatically analyze their models may not be able to use a given, specialized query approach if it does not fit their use case or modeling language. Furthermore, for reasons of economy and internal conventions, they may not be willing to change their preferred modeling language for every new analysis purpose to render some specialized analysis approach applicable [Aa13].

A previous utility evaluation of a multi-purpose and language independent model query language [De10] revealed that process managers perceive such a mechanism to be highly beneficial to support model analysis [Be11]. The query language presented in this study, however, uses formal set operations to define pattern queries. We argue that the ease of use of this language is rather low as pattern specification is cumbersome (cf. application examples in [Be11]). We argue that a query language is much easier to use if it allows for graphically modeling a pattern in much the same way as a model is developed.

In this paper, we thus present a multi-purpose and modeling language independent model query approach that allows for specifying patterns graphically. The approach consists of a graphical pattern editor and a matching algorithm. The theoretical foundations of our approach are based on graph theory, because any conceptual model is essentially a graph consisting of nodes and edges, no matter what modeling language is used. These nodes and edges are usually attributed with a type and a label, sometimes multiple labels, such as a name, cost, time, etc. A BPMN model [BPM13], for instance, contains nodes of type "task" and may have a name containing the value "grant credit".

In graph theory, the problem of pattern matching is known as the problem of subgraph isomorphism [Ul78]. Corresponding algorithms are able to detect exact pattern matches (i.e., the pattern match and the pattern have to be structurally and semantically identical). In the context of analyzing conceptual models however, it is often necessary to identify paths of model elements that are of previously unknown length (cf. requirements described in Section 2). Consequently, a matching mechanism is required that allows for identifying subgraphs in a model that are not strictly identical to a predefined pattern, but may contain paths of previously unknown length. In graph theory, this kind of pattern matching is known as the problem of subgraph homeomorphism [LW09]. Unfortunately, corresponding algorithms produce a huge number of pattern matches because by default all pattern edges are mapped to paths of all possible lengths in the model. This, however, is often not necessary, because only few pattern edges may need to be mapped to paths in the model. The resulting huge number of potentially irrelevant matching results leads to increased runtimes [LW09]. We therefore introduce a new type of graph problem that we call relaxed subgraph isomorphism, in which a node in the pattern graph has exactly one equivalent node in the model, but an edge in the pattern graph may – if so specified – be mapped to a path of elements in the model.

The purpose of this paper is thus to formally specify, implement, and evaluate a graph-based query approach for conceptual models using relaxed subgraph isomorphism detection. It takes into account that nodes and edges of conceptual model graphs can be annotated with various attributes as described above and that these attributes may have to be checked within the pattern matching process. Furthermore, we take into account that conceptual models may contain both directed and undirected edges, as well as more than one edge between two given nodes (e.g., hierarchy structures in Entity-Relationship Models (ERM), [Ch76]). Summarizing, the contribution of this paper is as follows:

- From a theoretical point of view, we introduce a new type of graph problem called relaxed subgraph isomorphism designed to address the particular requirements of pattern matching in conceptual models. We furthermore present a novel approach to

solve this problem, including specifics of conceptual models such as mixed directed/undirected edges as well as node and edge semantics. Up to now, only related work on subgraph isomorphism and subgraph homeomorphism exists. It does not take into account configurable edge-path mappings as described above. Also, it is restricted to either undirected or directed edges and mostly ignores node and edge semantics, which are both of utmost importance when analyzing conceptual models.
- From a practical point of view, the query approach supports a wide variety of different model analysis objectives involving pattern matching (cf. examples above). It is thus not restricted to supporting one objective alone.
- The query language is furthermore not restricted to finding pattern matches in conceptual models of a particular type or modeling language, but can be used on models of any type or modeling notation – as long as they are based on graphs.
- The query language allows to graphically model a pattern in much the same way as a model is developed. We argue that it is thus easier to use than query languages that rely on text-based pattern specification and follow the evaluation results of [Be11].

The remainder of the paper is structured as follows: In Section 2, we formulate functional requirements for pattern matching in conceptual models that were derived from typical patterns coming from literature on model analysis. In Section 3, we formally introduce the notions of a conceptual model, of subgraph isomorphism, and of relaxed subgraph isomorphism as a basis for the matching process. In a next step, we describe the graphical specification of patterns and briefly outline how our matching algorithm realizes relaxed subgraph isomorphism (Section 4). Section 5 presents a prototypical implementation of the model query approach using a meta-modeling tool. In Section 6, we evaluate our solution by arguing for its utility in comparison to existing model analysis approaches. The paper concludes with a summary of its contribution, limitations and an outlook to future research in Section 7.

2 Requirements of Pattern Matching in Conceptual Models

Several patterns that are relevant for model analysis purposes have been discussed in the literature and stem from research fields already named in the introduction. In particular, in order to identify model sections that match such patterns, a model query approach should comply with the following requirements (for a detailed discussion related to these requirements and an empirical derivation cf. [De14]):

- Requirement 1 (R1): The matching algorithm of the query approach should be able to return model subgraphs that structurally exactly match a predefined pattern graph. For example, this is needed to identify neighbored model elements, as it is necessary, for instance, for syntax checking (e.g., if two nodes of different types are not allowed to be connected by edges).
- Requirement 2 (R2): The matching algorithm of the query approach should be able to return model elements that have a particular type like “BPMN task”, “EPC function”, etc., whenever this is specified in the pattern. If such types are not specified, elements of any type should be returned. This is necessary for all of the business tasks

outlined in Section 1. For instance, for compliance checking, we need to know if two activities follow each other in a process model.

- Requirement 3 (R3): The matching algorithm of the approach should be able to return model elements that have a particular label like "Check invoice" whenever this is specified in the pattern. If such labels are not specified, the algorithm should return elements with any label. For instance, we need to be able to evaluate labels if we search for weaknesses (e.g., a "print" activity followed by a "scan" activity).
- Requirement 4 (R4): The matching algorithm of the query approach should be able to return model fragments containing a path of previously unknown length. This means that it should be possible to specify a pattern that contains simple edges and edges that are mapped to a path in the model. For example, compliance incorporates rules that prescribe that before an activity is performed in a process, another one must have been performed before, but it is not necessary that these both activities directly follow each other. So, we must be able to check whether there is a path between them.

These requirements have been derived from the structure and the label semantics such patterns typically have. In the following, we provide a brief example for each requirement. For a comprehensive list of patterns, we refer to the literature (e.g., [Bec10], [Ga08], [ODA08], [Me08], [Be11], [ADW08], [Me07], [PGD12], [Ba05], and [De14]). In addition to these requirements, we argue that a query language for conceptual models should be applicable to not only process models, but also to any other type of conceptual model like data models, organizational charts, ontologies, etc. It should furthermore not be limited to analyzing models developed in a particular modeling language [Aa13]. Also, the approach should provide a graphical model editor (cf. [SA13] for an additional discussion on the benefit of graphical pattern editors compared to text-based approaches). We therefore add two additional requirements as follows:

- Requirement 5 (R5): The query language should be applicable to conceptual models of multiple types and languages.
- Requirement 6 (R6): The query language should provide a graphical pattern editor that allows for modeling a pattern graph according to R1 to R5. This pattern graph is then augmented to include modeling language-specific type and label information.

3 Formal Specification

In this section, we formalize the functional requirements identified above. To do so, we first introduce the concepts of a conceptual model in terms of a graph and then proceed with defining the problem of subgraph isomorphism (cf. R1). Finally, we note that subgraph isomorphism must be relaxed to account for the requirements we identified (cf. R4). To account for the variety of conceptual modeling languages, we keep our definition of a conceptual model as abstract as possible. The goal is to ensure that the model query language can be used flexibly, no matter what modeling language is used or in which way a language was adapted (cf. R5). We assume that conceptual models consist of nodes representing any domain object of interest, and of edges describing relationships between them. Additional information regarding the nature of objects and relationships are conceptualized as attributes annotated to nodes and edges (cf. R2, R3).

Definition 1 (conceptual model): A conceptual model is a tuple $M=(O,R,A,\alpha)$, with O being a non-empty set of objects (nodes) and R being a non-empty set of relationships (edges). We write $E=O \cup R$ to denote the set of all model elements. $R=R_D \cup R_U$ consists of directed and undirected relationships. As in many modeling languages, multiple relationships are allowed between the same two nodes (like, e.g., hierarchy structures in ERMs), we use *multi-edges* to define relationships as follows: $R_D \subseteq O \times O \times \boldsymbol{N}$ is the set of directed relationships between the objects of a conceptual model (i.e., directed edges of the model graph). $R_U \subseteq \{\{o_x,o_y,n\} \mid o_x,o_y \in O,\ n \in \boldsymbol{N},\ o_x \neq o_y\}$ is the set of undirected relationships between the objects of a conceptual model (i.e., undirected edges of the model graph). $\boldsymbol{N}$ is the set of natural numbers used to number multi-edges. Numbering of multi-edges always starts at $n=1$ and increases by one for each additional edge. A is a non-empty-set of attributes carrying all information that can be associated with elements of E. It can be used, for example, to assign an element a type (e.g., a BPMN "task" or an EPC "function", cf. R2) or a label (e.g., "receive goods" to describe an activity in a process, cf. R3). In general, elements from A can be high-dimensional vectors of attributes assigned to either objects or relations via the function $\alpha: E \rightarrow A$.

As the goal of the model query language is to map elements of one conceptual model (a pattern) to those of another one (the model), we must define which elements are compatible with each other. Clearly, this depends on the context of application (cf. Section 2). It may be a simple equality check of element types (cf. R2) or a full-fledged check applying similarity measures to textual descriptions (cf. R3). Again, to keep things general, we define only an equivalence relation $\sim \subseteq A \times A$ on attributes. As an example, this equivalence relation could be implemented as a simple equality check on types. More complicated equivalence relations taking into account multiple attribute dimensions and based on, for instance, string similarity, linguistic [DHL09] or ontological [TF09] similarity measures are easily conceivable.

We use the notation $M' \leq M$ to denote that M' is a model that can be obtained from M by removing objects from O, relationships from R, and by reducing the domain of α accordingly. Formally, this reduction can be described as $O' \subseteq O$, $R' \subseteq R$, and $\alpha'=\alpha|_{E'}$. In terms of graph theory, M' is called a subgraph of M and exactly matches a fragment of M. We define a pattern query P as a model that is searched for in another one M ($|O_P| \leq |O_M|$ and $|R_P| \leq |R_M|$). Given a model graph M and a pattern graph P, P is isomorphic to a subgraph $M' \leq M$ if there is a structure preserving one-to-one mapping φ between all elements of P and all elements of M' (cf. R1) satisfying the equivalence relation $\alpha(e_P) \sim \alpha(\varphi(e_P))$. The following definition formalizes this subgraph isomorphism relation.

Definition 2 (subgraph isomorphism): Given a pattern graph $P=(O_P,R_P,A,\alpha_P)$, a model graph $M=(O_M,R_M,A,\alpha_M)$, and an equivalence relation $\sim \subseteq A \times A$, P is subgraph-isomorphic to M if and only if it exists an $M' \subseteq M$ such that there exists a bijection $\varphi: E_P \rightarrow E_{M'}$ satisfying the isomorphism condition:

$$(o_x,o_y,n_P) \in R_P \Leftrightarrow (\varphi(o_x),\varphi(o_y),n_M) \in R_{M'};\ \{o_x,o_y,n_P\} \in R_P \Leftrightarrow \{\varphi(o_x),\varphi(o_y),n_M\} \in R_{M'};$$
$$\alpha(o_{Pj}) \sim \alpha(\varphi(o_{Pj})),\ \alpha(r_{Pj}) \sim \alpha(\varphi(r_{Pj})),\ n_P,n_M \in \boldsymbol{N}$$

To extend this definition towards relaxed subgraph isomorphism, we first have to introduce the notion of a path. A path in a model graph M can be understood as a sequence of objects $<o_1,...,o_n>$ such that $(o_i,o_{i+1},z_i) \in R \vee (o_{i+1},o_i,z_i) \in R \vee \{o_i,o_{i+1},z_i\} \in R \ \forall i \in \{1..n-1\}, z_i \in \mathbf{N}$. We write $paths(o_x,o_y)$ to denote the set of all paths between nodes o_x and o_y. Special types of paths are those obeying certain constraints on the directions of their edges. $paths_d(o_x,o_y)$ shall denote the set of all directed paths from o_x to o_y, meaning all sequences of objects $<o_x,...,o_y>$ such that $(o_i,o_{i+1},z_i) \in R \ \forall i \in \{x..y-1\}, z_i \in \mathbf{N}$. Conversely, $paths_u(o_x,o_y)$ shall denote the set of all undirected paths between o_x to o_y, meaning all sequences of objects $<o_x,...,o_y>$ such that $\{o_i,o_{i+1},z_i\} \in R \ \forall i \in \{x..y-1\}, z_i \in \mathbf{N}$. Finally, $paths_a(o_x,o_y)$ shall denote the set of all paths between o_x and o_y, ignoring the direction of the contained edges, that is, all sequences of objects $<o_x,...,o_y>$ such that $\{o_i,o_{i+1},z_i\} \in R \vee (o_i,o_{i+1},z_i) \in R \vee (o_{i+1},o_i,z_i) \in R \ \forall i \in \{x..y-1\}, z_i \in \mathbf{N}$.

Based on paths, we define the notion of relaxed subgraph isomorphism (cf. R4). In the subgraph isomorphism definition, any pair of model nodes must be connected directly via an edge (i.e., a path of length one) whenever the two pattern graph nodes that map on them are connected by an edge. In the relaxed notion, these model graph nodes may be connected via paths of any length. Effectively, edges in the pattern graph *can* be mapped to paths instead of edges in the model graph, but only when this is explicitly specified in the pattern. To choose for which pattern edges this generalization shall be used, and which kind of path definition should be applied, we define a function $p: R_P \rightarrow \{edge, normalPath, mixedPath\}$. It indicates whether an edge of the pattern graph shall correspond to an edge in the model graph (edge), a directed or undirected path (normalPath), or any path regardless of the directions of its edges (mixedPath). The value must be specified by the user for each pattern edge. In the following, we call edges of the pattern to be mapped to paths in the model *path-edges*.

Definition 3 (relaxed subgraph isomorphism): Given a pattern graph $P=(O_P,R_P,A,\alpha_P)$, a model graph $M=(O_M,R_M,A,\alpha_M)$, an equivalence relation $\sim \subseteq A \times A$, and a function $p: R_P \rightarrow \{edge, normalPath, mixedPath\}$, P is relaxedly subgraph-isomorphic to M if and only if it exists an $M' \subseteq M$ such that there exists a left-total relation $\psi: E_P \rightarrow \mathbf{P}(E_{M'})$ satisfying the relaxed isomorphism condition:

$$(o_x,o_y,n_P) \in R_P \wedge p((o_x,o_y,n_P))=edge \Leftrightarrow (\psi(o_x),\psi(o_y),n_M) \in R_{M'}, \alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})),$$
$$\alpha(r_{Pj}) \sim \alpha(\psi(r_{Pj})), n_P,n_M \in \mathbf{N}; \{o_x,o_y,n_P\} \in R_P \wedge p(\{o_x,o_y,n_P\})=edge \Leftrightarrow \{\psi(o_x),\psi(o_y),n_M\} \in R_{M'},$$
$$\alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})), \alpha(r_{Pj}) \sim \alpha(\psi(r_{Pj})), n_P,n_M \in \mathbf{N}; (o_x,o_y,n_P) \in R_P \wedge p((o_x,o_y,n_P))=normalPath$$
$$\Leftrightarrow |paths_d(\psi(o_x),\psi(o_y))|=max(n_P), \alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})), n_P \in \mathbf{N}; \{o_x,o_y,n_P\} \in R_P \wedge$$
$$p(\{o_x,o_y,n_P\})=normalPath \Leftrightarrow |paths_u(\psi(o_x),\psi(o_y))|=max(n_P), \alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})), n_P \in \mathbf{N};$$
$$(o_x,o_y,n_P) \in R_P \wedge p((o_x,o_y,n_P))=mixedPath \Leftrightarrow |paths_a(\psi(o_x),\psi(o_y))|=max(n_P),$$
$$\alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})), n_P \in \mathbf{N}; \{o_x,o_y,n_P\} \in R_P \wedge p(\{o_x,o_y,n_P\})=mixedPath \Leftrightarrow$$
$$|paths_a(\psi(o_x),\psi(o_y))|=max(n_P), \alpha(o_{Pj}) \sim \alpha(\psi(o_{Pj})), n_P \in \mathbf{N}$$

The definition assures that, in one mapping, each node of the pattern is matched to exactly one node in the model, every simple edge of the pattern is matched exactly to one edge in the model, and every path-edge of the pattern is matched to exactly one path in the model. Whenever two nodes of the pattern are connected by more than one (n) path-

edge, the definition assures that these *n* path-edges are always mapped to *n different* paths in the model (cf. the $max(n_P)$ condition).

4 The Query Language

Based on the definitions above, pattern matching always starts with the definition of a pattern that should be searched for in conceptual models. To that end, we introduce the syntax of a pattern query. A pattern query is essentially a graph consisting of nodes and edges as described in *Definition 1*. To define a pattern query, we propose not to have it specified formally, but have it drawn and transformed automatically into a formal representation complying with the pattern definition afterwards. For example, consider the pattern query shown in Figure 1. It represents a behavioral conflict in EPCs. In particular, when the triggering event fires, the succeeding AND connector may never fire when the process instance was routed to the other path by the XOR connector [Me07].

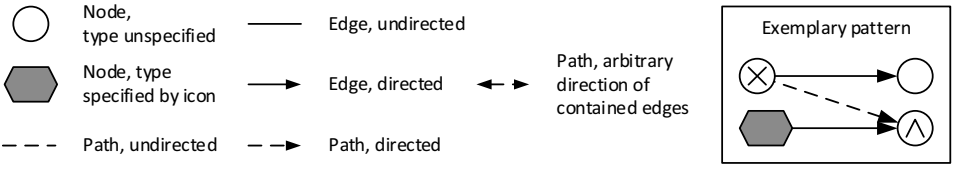

Figure 1: Pattern specification in the graphical concrete syntax

Nodes o_P of such a visual pattern query are represented by circles and edges/paths r_P by lines. If a node is assigned a node type (e.g., "entity type") as part of $\alpha(o_P)$, we propose to attach an icon according to the representation of a corresponding node type in a model. In the example, three nodes are typed, so they appear as a red hexagon, a circle containing an "x" and a circle containing a "∧", standing for "event", "XOR connector", and "AND connector". One node is not typed, so it can be mapped to any node of a model. Labels as further parts of $\alpha(o_P)$ should appear within the boundary of a node. Further attributes of $\alpha(o_P)$ should not appear as a visual part of the pattern query. In an implementation, they should be specified by opening a context menu. Pattern edges to be mapped to edges in a model r_P are represented by solid lines with one attached arrow if they are directed. Pattern edges to be mapped to paths in a model r_P are represented by dashed lines with an arrow attached if they are directed. Without an arrow, they are undirected. With arrows at both ends, they should be mapped to paths containing edges of any direction. As we cannot identify the type of an edge only from its representation in many modeling languages, we propose to attach the type of the edge – if specified – textually. Any other attributes of $\alpha(r_P)$ should be handled like those of $\alpha(o_P)$.

5 Implementation

To demonstrate the feasibility of our model query language, we implemented it as a plugin for a meta-modeling tool that was available from a previous research project.

Being a meta-modeling tool, it allows for specifying modeling languages at runtime. Conceptually, it is based on the idea that any modeling language essentially provides a set of object types and relationship types. To create a model these types are instantiated into concrete objects and relationships. The query language we propose is thus based on the same constructs that are used on meta-level to define a modeling language. This is why the query language is essentially modeling language-independent.

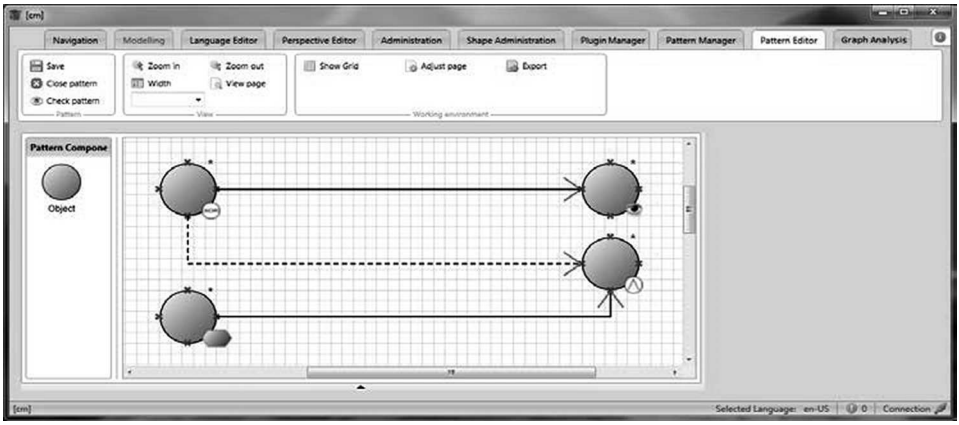

Figure 2: Pattern editor

The implementation of the query language includes a pattern editor as depicted in Figure 2. This editor allows for drawing a pattern graph complying with the Requirements R1 to R5 derived above (R6). The analyst can specify a name for the pattern query and choose the modeling language the pattern query is supposed to be valid for (R5). The type of a node or edge can be customized according to the corresponding modeling language. The plugin accesses the modeling language specification in order to get all type information that is required during the matching process. The exemplary pattern query in Figure 2 matches the example in Figure 1.

After having specified a pattern query or choosing a previously specified one, the user can specify which models should be analyzed. We implemented an algorithm that matches the pattern to the models according to the formal definitions in Section 3. The algorithm determines all pattern matches in all input models. The plugin returns a list of models that contain pattern matches. By selecting an entry from this list, the model is loaded in the modeling environment of the meta-modeling tool. All returned pattern matches are highlighted in different colors to allow for retracing which pattern node was mapped to which model node and which pattern edge was mapped to which model edge or path (cf. Figure 3). In the example, a pattern match was found that contains a path between the XOR split and the AND join. If a model contains more than one pattern match, the user can browse through the matches, meaning that the highlighting of the model changes to the corresponding places for every match.

To assure the applicability of the model query approach and its implementation, we performed a preliminary runtime experiment, in which we searched for fourteen specific

patterns. Seven of these patterns were EPC patterns, and seven were ERM patterns. We applied them to 53 EPCs (sizes from 15 to 294 elements) and 42 ERMs (sizes from 16 to 97 elements) coming from the retail industry (for details on the models and patterns, please see [De14], where we used the same patterns and models to test another query language). We conducted the performance evaluation on an Intel® Core™ 2 Duo CPU E8400 3.0 GHZ with 4 GB RAM and Windows 7 (62-Bit edition). We disabled the energy saving settings in Windows and executed the application as a 32-bit real-time process to avoid any unnecessary hardware slow down or process switching. As a result, we observed runtimes for searching one pattern in one model ranging from fractions of a millisecond to just under five seconds. In more than 90% of the pattern matching cases, results could be obtained in less than 100 milliseconds for ERMs (in less than 10 milliseconds for EPCs). Note that the (few) long runtimes were due to path searches with most of the restrictions like type, label etc. turned off. For instance, one ERM search that took 2130 milliseconds returned 22856 matches. Hence, we evaluate the overall performance of the approach satisfactory.

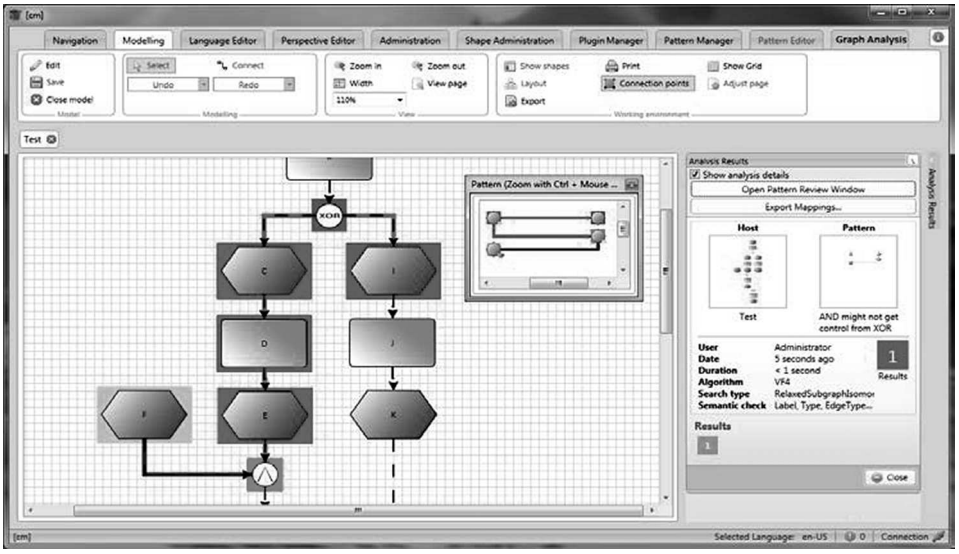

Figure 3: Pattern match visualization

6 Related Work

Table contains a detailed comparison of our work to other model query languages proposed in the literature. We draw on the requirements for a graph-based model query language presented in Section 2 to compare our work with existing approaches. A query language thus has to be able to find arbitrary isomorphic substructures in a model graph (R1), consider type (R2) and (R3) label information in its matching process and find paths of arbitrary length (R4). A query language should furthermore support querying conceptual models of any type or graph-based modeling language (R5) and provide a

graphical pattern editor (R6). The table contains a "+" if the query language fulfills a given requirement and a "-" otherwise. The table contains a "0" if the given requirement is only partly fulfilled (see details below).

Three classes of query languages can be distinguished. *The first class* contains all process model query languages. aFSA [MW06], APQL [So11], BeehiveZ [Ji10], BPMN VQL [FT09], BPMN-Q [Aw07], BPQL [MS04], BP-QL [Be08], IPM-PQL [CKJ07], PPSL [Fö07] and an approach based on indexing and untanglings [PRH14] belong to this class. These query languages can only be applied to process models. Some of these languages were designed to query models of a particular process modeling language. BPMN-Q [Aw07] is a prominent example. Other authors define a new process modeling language and propose a query approach for this language (cf. BPQL and BP-QL). The *second class* of query languages contains those approaches that are specific to UML models. EMF-IncQuery [Ber10] and OCL [OCL13] fall into this category. As with process model query languages, these approaches are thus designed to query models of a particular type or modeling languages. Pattern queries are created by means of a declarative programming language. The *third class* contains general graph query languages. SPARQL [W3C13] as well as the Neo4j query language Cypher [Ne13] are prominent examples that fall into this category. They provide functionality that is similar to the feature set of our query language. These approaches are essentially declarative programming languages. A graphical pattern editor is not provided.

Query Language / Requirement	R1	R2	R3	R4	R5	R6
aFSA	+	+	+	-	-	+
APQL	+	+	+	+	-	-
BeehiveZ	+	+	+	-	-	+
BPMN VQL	+	+	+	+	-	+
BPMN-Q	+	+	+	+	-	+
BPQL	+	+	+	-	-	-
BP-QL	+	+	+	+	-	+
Cypher (Neo4j)	+	+	+	+	+	-
EMF-IncQuery	+	+	+	+	-	-
IPM-PQL	+	+	+	+	-	-
OCL	+	+	+	-	-	-
PPSL	+	+	+	+	-	+
SPARQL	+	+	+	+	+	-
Untanglings	-	0	+	+	0	-
VMQL	+	+	+	+	0	0
Our work	+	+	+	+	+	+

Table 1: Comparison of our work to other model query languages

VMQL [St11] is a visual model query language providing functionality that is similar to that of our approach. The language is also intended to be applicable on models developed in any modeling language. However, this has been demonstrated for UML as well as BPMN models only (i.e., R5="0") [SA13]. Extending VMQL to additional modeling languages however requires developing a new pattern editor that provides the element types of that particular modeling language (i.e., R6="0").

7 Contributions, limitations, and outlook

The purpose of this paper was to introduce a query language for conceptual models that is applicable to models of any type or modeling language. Our work can furthermore support multiple business objectives of analyzing models and is thus expected to be broadly applicable. Hence, the contribution of the paper can be summarized as follows:

- The query language we propose includes a matching algorithm that is based on a new kind of graph search problem called *relaxed subgraph isomorphism*, which is particularly related to the analysis of conceptual models. Therefore, we contribute to algorithmic graph theory by introducing a new class of graph search problem. With our work, we expect to trigger further research on this problem from a graph theoretical perspective. For instance, novel efficiency-enhanced algorithms would contribute both to graph theory and – in turn – to conceptual model analysis.
- In practice, only few multi-purpose and language-independent model query languages exist up to now. With our work we hence contribute to a variety of different model analysis objectives requiring pattern matching. Furthermore, our approach can be used on models of any graph-based modeling language. Therefore, we expect wide-spread application, as model analysis and model analysts are no longer restricted to particular modeling languages or few analysis objectives.
- Another important characteristic of our work is its graphical pattern editor easing the usage of the query language.

However, our work reveals some limitations resulting from its broad applicability:

- Defining pattern queries for a particular application scenario is the responsibility of an analyst. Furthermore, a pattern query has to be defined according to the modeling languages of the models to be analyzed. Naturally, although searching for ERM patterns in EPCs is possible using our query language, it will not return any results.
- Another restriction of our query language also results from its applicability to multiple modeling languages. It is by design not able to analyze special characteristics of special model types, for instance execution semantics of process models.
- Our approach has not yet been subject to a comprehensive utility evaluation. Despite this, we expect analysts to appreciate it, as related works have already proven to be highly relevant for model analysis purposes [Be11]. Moreover, due to the possibility to specify patterns graphically, we expect an even higher utility of our approach. Comprehensive utility evaluations are subject of short-term research. In particular, we plan to apply our work in financial institutions to check model repositories containing integrated business process models, data models and organizational charts for regulatory compliance violations and weaknesses. This will furthermore carve out additional functional requirements that have to be included in the query language.

Although we already conducted performance experiments suggesting satisfactory runtime of the query approach (not included in this paper), medium-term research will focus on performing additional runtime experiments on extremely large models with the aim of further increasing execution speed and applying the algorithm to further model analysis scenarios in practice. At the moment, we investigate graph-theoretical structural

characteristics of conceptual models that can speed up pattern matching. In particular, we expect bounded treewidth and planarity of conceptual model graphs to be very promising characteristics to increase matching performance significantly.

References

[Aa13] van der Aalst, W.M.P.: Business Process Management: A Comprehensive Survey. ISRN Software Engineering. 2013 (2013), pp. 1-37.

[ADW08] Awad, A.; Decker, G.; Weske, M: Efficient compliance checking using bpmn-q and temporal logic. In: Business Process Management. Berlin 2008, pp. 326-341.

[Aw07] Awad, A.: BPMN-Q: A Language to Query Business Processes. In: Proceedings of the EMISA 2007, pp. 115-128.

[Ba05] Batra, D.: Conceptual data modeling patterns: Representation and validation. Journal of Database Management (JDM), 16 (2005) 2, pp. 84-106.

[Be08] Beeri, C.; Eyal, A.; Kamenkovich, S.; Milo, T.: Querying business processes with BP-QL. Information Systems, 33 (2008) 6, pp. 477-507.

[Be11] Becker, J.; Bergener, P.; Delfmann, P.; Weiss, B. (2011). Modeling and Checking Business Process Compliance Rules in the Financial Sector. In: Proceedings of the ICIS 2011.

[Bec10] Becker, J.; Bergener, P.; Räckers, M.; Weiß, B.; Winkelmann, A.: Pattern-Based Semi-Automatic Analysis of Weaknesses in Semantic Business Process Models in the Banking Sector. In: Proceedings of the ECIS 2010. Pretoria 2010.

[Ber10] Bergmann, G.; Horváth, Á.; Ráth, I.; Varró, D.; Balogh, A.; Balogh, Z.; Ökrös, A.: Incremental Evaluation of Model Queries over EMF Models. In: Proceedings of the International Conference MODELS 2010. Oslo 2010, pp. 76-90.

[BPM13] Object Management Group, Business Process Model and Notation 2.0, 2013. (http://www.omg.org/spec/BPMN/2.0/).

[Ch76] Chen, P.P.-S.: The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems. 1 (1976) 1, pp. 9-36.

[CKJ07] Choi, I.; Kim, K.; Jang, M.: An XML-Based Process Repository and Process Query Language for Integrated Process Management. Knowledge and Process Management 14 (2007) 4, pp. 303-316.

[De10] Delfmann, P.; Herwig, S.; Lis, L.; Stein, A.; Tent, K.; Becker, J.: Pattern Specification and Matching in Conceptual Models. A Generic Approach Based on Set Operations. Enterprise Modelling and Information Systems Architectures 5 (2010) 3, S. 24-43.

[De14] Delfmann, P.; Steinhorst, M.; Dietrich, H.-A.; Becker, J.: The Generic Model Query Language GMQL – Conceptual Specification, Implementation, and Runtime Evaluation. Information Systems. Accepted for publication, DOI: 10.1016/j.is.2014.06.003.

[Di11] Dijkman, R.; Dumas, M.; Van Dongen, B.; Käärik, R.; Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems. 36 (2011) 2, pp. 498-516.

[DHL09] Delfmann, P.; Herwig, S.; Lis, L.: Unified Enterprise Knowledge Representation with Conceptual Models – Capturing Corporate Language in Naming Conventions. In: Proceedings of the ICIS 2009. Phoenix 2009.

[DRR12] Dijkman, R. M.; La Rosa, M.; Reijers, H. A.: Managing Large Collections of Business Process Models – Current Techniques and Challenges. Computers in Industry 63 (2012) 2, pp. 91-97.

[Fö07] Förster, A.; Engels, G.; Schattkowsky, T.; Van Der Straeten, R.: Verification of business process quality constraints based on visual process patterns. In: Proceeding sof the 1st

Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering 2007, pp. 197-208.

[FT09] Di Francescomarino, C.; Tonella, P.: Crosscutting concern documentation by visual query of business processes. In: BPM Workshops 2009. Berlin 2009, pp. 18-31.

[Ga08] García-Bañuelos, L.: Pattern Identification and Classification in the Translation from BPMN to BPEL. In: On the Move to Meaningful Internet Systems: OTM 2008. Berlin 2008, pp. 436-444.

[Ji10] Jin, T.; Wang, J.; Wu, N.; La Rosa, M.; Ter Hofstede, A. H. (2010). Efficient and accurate retrieval of business process models through indexing. In: On the Move to Meaningful Internet Systems: OTM 2010. Berlin 2010, pp. 402-409.

[Kn10] Knuplesch, D.; Ly, L. T.; Rinderle-Ma, S.; Pfeifer, H.; Dadam, P.: On Enabling Data-Aware Compliance Checking of Business Process Models. In: Proceedings of the 29th International Conference on Conceptual Modeling (ER 2010). Vancouver 2010, pp. 332-346.

[LW09] Lingas, A.; Wahlen, M.: An exact algorithm for subgraph homeomorphism. Journal of Discrete Algorithms 7 (2009) 4, pp. 464-468.

[Me07] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models, Vienna University of Economics and Business Administration. Vienna 2007.

[Me08] Mendling, J.; Verbeek, H. M. W.; van Dongen, B. F.; van der Aalst, W. M. P.; Neumann, G.: Detection and prediction of errors in EPCs of the SAP reference model. Data & Knowledge Engineering 64 (2008) 1, pp. 312-329.

[MS04] Momotko, M.; Subieta, K.: Process Query Language: A Way to Make Workflow Processes More Flexible. In: Proceedings of the 8th East European Conference on Advances in Databases and Information Systems (ADBIS 2004). Budapest 2004, pp. 306-321.

[MW06] Mahleko, B.; Wombacher, A.: Indexing Business Processes based on Annotated Finite State Automata. In: Proceedings of the 2006 IEEE International Conference on Web Services (ICWS'06). Washington, DC 2006, pp. 303-311.

[Ne13] Neo4j: Cyper Query Language. 2013. (http://docs.neo4j.org/chunked/stable/cypher-query-lang.html).

[OCL13] Object Management Group: Object Constraint Language, 2013. (http://www.omg.org/spec/OCL/2.3.1/).

[ODA08] Ouyang, C.; Dumas, M.; Van Der Aalst, W.M.P.: Pattern-based translation of BPMN process models to BPEL web services. International Journal of Web Services Research (IJWSR), 5 (2008) 1, pp. 42-62.

[PGD12] Polyvyanyy, A.; García-Bañuelos, L.; Dumas, M.: Structuring acyclic process models. Information Systems 37 (2012) 6, pp. 518-538.

[PRH14] Polyvyanyy, A.; La Rosa, M.; ter Hofstede, A. H. M.: Indexing and Efficient Instance-based Retrieval of Process Models Using Untanglings. In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014). Thessaloniki, Greece, pp. 439-456.

[SA13] Störrle, H.; Acretoaie, V.: Querying business process models with VMQL. In: Proceedings of the 5th ACM SIGCHI Annual International Workshop on Behaviour Modelling – Foundations and Applications (BMFA 2013). New York 2013, pp. 1-10.

[So11] Song, L.; Jianmin, W.; ter Hofstede, A. H. M.; La Rosa, M.; Ouyang, C.; Wen, L.: A Semantics-based Approach to Querying Process Model Repositories. QUT eprints 2011.

[St11] Störrle, H.: VMQL: A visual language for ad-hoc model querying, Journal of Visual Languages & Computing. 22 (2011) 1, pp. 3-29.

[TF09] Thomas, O.; M. Fellmann M.: Semantic Process Modeling – Design and Implementation of an Ontology-based Representation of Business Processes. Business & Information Systems Engineering. 1 (2009) 6, pp. 438-451.

[Ul78] Ullmann, J. R.: An Algorithm for Subgraph Isomorphism. Journal of the ACM 23 (1976) 1, pp. 31-42.

[W3C13] World Wide Web Consortium: SPARQL Query Language for RDF, 2013. (http://www.w3.org/TR/rdf-sparql-query/).
[YDG12] Yan, Z.; Dijkman, R.; Grefen, P.: Fast business process similarity search. Distributed and Parallel Databases, 30 (2012) 2, pp. 105-144.